GUIDE DE L'ORIC

Philippe Bayvejiel

Edimicro

GUIDE DE L'ORIC

Philippe BAYVEJIEL est le pseudonyme des quatre auteurs de l'ouvrage : Michel BIRADES, Dominique JOLY, Philippe LE BITOUX, Jean-Marie VALLADE.

Cet ouvrage a été réalisé sous la direction de Benoît de MERLY.

© F.D.S./Edimicro 1983
Première édition

Imprimé en France. Droits mondiaux réservés.

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article 40) ».
« Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal ».

ISBN : 2-904457-02-X

EDIMICRO

DÉPARTEMENT ÉDITIONS DE F.D.S. SARL

121-127, avenue d'Italie, 75013 Paris. Tél. 585.00.00

GUIDE DE L'ORIC

par

Philippe BAYVEJIEL

Edimicro

Préface

Vous êtes ou vous serez bientôt, en possession d'un microordinateur Oric. Votre achat est justifié par des besoins personnels (curiosité, besoin de développer l'informatique familiale) ou professionnels (problèmes de paye, de comptabilité, de gestion de stocks).

Vous désirez maintenant apprendre à vous servir de votre machine, c'est-à-dire découvrir et utiliser au mieux toutes ses possibilités, et ainsi rentabiliser votre nouvelle acquisition. C'est là le but de ce guide.

A qui s'adresse le GUIDE DE L'ORIC ? Principalement à ceux qui n'ont que des connaissances très restreintes en informatique ou en programmation. Il s'agit donc avant tout d'une initiation, mais d'une initiation suffisamment poussée, pour vous permettre de programmer correctement.

Toutes les notions dont vous avez besoin sont abordées progressivement. A l'aide de programmes simples et commentés, vous assimilerez sans peine le maniement de la machine et vous maîtriserez le vocabulaire de base pour converser avec votre Oric. Ce vocabulaire sera ensuite approfondi pour vous permettre de faire des graphiques, de la musique ou des mathématiques.

Nous vous engageons, lors de votre apprentissage, à faire exécuter par Oric les exemples commentés, puis les modifier au gré de votre imagination. En effet, la pratique de l'ordinateur est, avant tout, expérimentale. Seule une utilisation effective vous permettra de progresser rapidement et d'acquérir les quelques règles simples qui feront de vous un bon utilisateur.

A l'issue de la lecture de ce livre, vous aurez constaté que l'ensemble des possibilités de votre Oric n'a pas d'autres limites que celles de votre imagination...

Table des matières

Chapitre 1 : Présentation générale de l'Oric 1

1.1 Introduction 1

1.2 Le clavier 2

1.3 L'unité centrale 4

1.4 La sortie vidéo 7

1.5 La sortie cassette 7

1.6 Les extensions 8

1.7 Mise en route 8

Chapitre 2 : Initiation à la programmation 11

2.1 Introduction 11

2.1.1 Stockage de l'information 13
2.1.2 Notion de langage - Le Basic 15

2.2 Mode direct - Mode différé 16

2.2.1 Introduction 16
2.2.2 Fonctionnement en mode direct 18
- Les opérations élémentaires
* addition
* soustraction
* multiplication
* division
* puissance
- Les ordres de priorité des opérateurs
- Notion de variable

2.3 Les bases de la programmation - Découverte du langage Basic 26

2.3.1 Le premier programme 26
2.3.2 L'instruction GOTO 29
2.3.3 La commande LIST 30

2.3.4 CTRL-C, CONT 31
2.3.5 RESET 32
2.3.6 L'instruction INPUT 33
2.3.7 L'instruction PRINT 34

2.4 Commandes fondamentales 36

2.4.1 Connaître le contenu d'un programme - La commande LIST 36
2.4.2 Effacement d'un programme - La commande NEW 38
2.4.3 Lancement d'un programme - La commande RUN 39
2.4.4 Fin d'un programme - Les instructions STOP/END 40
2.4.5 Corrections et modifications de programmes 40
- insertion d'une ligne
- suppression d'une ligne
- modification complète d'une ligne
- modification de quelques caractères sur une ligne

2.5 Créez votre jeu en Basic 42

2.5.1 Présentation du jeu 42
2.5.2 Instruction IF ... THEN ... ELSE 43
2.5.3 Première version du jeu 46
2.5.4 Fonction RND 47
2.5.5 Fonction INT 48
2.5.6 L'instruction REPEAT ... UNTIL 49
2.5.7 Seconde version du jeu 50
2.5.8 L'instruction WAIT 50
2.5.9 L'instruction EXPLODE 50
2.5.10 Notion de boucle - FOR ... NEXT 51

2.6 Conclusion 52

Chapitre 3 : Le langage Basic de l'Oric 53

3.1 Généralités 53

3.2 Constantes - Variables - Indices 54

3.2.1 Les constantes 54
- les constantes numériques
- les constantes alphanumériques

3.2.2 Les variables simples 58
- les chaînes de caractères
- les variables entières
- les variables réelles
3.2.3 Les tableaux 64
- les tableaux de variables entières
- les tableaux de chaînes de caractères
- les tableaux de variables réelles

3.3 Les opérateurs 71

3.3.1 Les opérateurs arithmétiques 71
3.3.2 Les opérateurs relationnels 72
3.3.3 Les opérateurs logiques 73
- action de AND
- action de OR
- action de NOT
3.3.4 Règles d'évaluation des expressions 74

3.4 Les instructions Basic facultatives 75

3.4.1 CLEAR 75
3.4.2 DIM 76
3.4.3 END 77
3.4.4 Fonction FRE 77
3.4.5 Instruction LET 78
3.4.6 Arrêt STOP 78
3.4.7 Commentaires REM 79

3.5 Les instructions de transfert 79

3.5.1 Branchement inconditionnel 79
- GOTO
- ON ... GOTO
3.5.2 Transferts conditionnels 82
- IF ... THEN ... ELSE

3.6 Les boucles 83

3.6.1 Boucle FOR ... NEXT 84
3.6.2 Boucle REPEAT ... UNTIL 88

3.7 Les sous-programmes 89

3.7.1 Présentation - GOSUB - RETURN 89
3.7.2 ON ... GOSUB 91

3.7.3 Appel de sous-programmes assembleurs - CALL 92
3.7.4 Fonctions assembleurs DEF ... USR 92

3.8 Les fonctions 92

3.8.1 Fonctions numériques internes 92
- génération d'un nombre aléatoire
- fonctions mathématiques simples
- fonctions mathématiques complexes
3.8.2 Fonctions de traitement des chaînes de caractères 93
- conversion code ASCII/caractère
- conversion chiffre/lettre
- extraction/traitement de chaînes
3.8.3 Création de fonctions utilisateurs 95

3.9 Les entrées - sorties 97

3.9.1 Entrées de données - L'instruction INPUT 97
3.9.2 Saisie d'un caractère - L'instruction GET 98
3.9.3 Saisie au vol d'un caractère - la fonction KEY$ 98
3.9.4 Lecture des données dans le programme READ ... DATA 99
3.9.5 L'instruction RESTORE 99

3.10 Conclusions 100

Chapitre 4 : Possibilités graphiques de l'Oric 101

4.1 Introduction 101

4.2 Les modes graphiques 102

4.2.1 Les modes basse-résolution TEXT, LORES 0, LORES 1 102
4.2.2 Le mode haute-résolution HIRES 106

4.3 Les commandes graphiques 108

4.3.1 Commandes graphiques basse-résolution 108
- affichage d'un caractère PLOT
- lecture du caractère affiché SCRN

4.3.2 Commandes graphiques haute-résolution 114
- positionnement du curseur CURSET
- déplacement du curseur CURMOV
- tracé d'une droite DRAW
- utilisation de pointillés PATTERN
- affichage d'un caractère CHAR
- tracé d'un cercle CIRCLE
- obtention de la couleur d'un point POINT
- remplissage de lignes FILL
- effets spéciaux

4.4 Graphiques avancés 127

Chapitre 5 : Les sons et l'Oric 135

5.1 Introduction 135

5.2 Les sons prédéfinis 136

5.3 Les commandes de sons 136

5.3.1 SOUND (canal, période, volume) 137
- le canal
- la période
- le volume

5.3.2 MUSIC (canal, octave, note, volume) 138
- le canal
- l'octave
- la note
- le volume

5.3.3 PLAY (combinaison des canaux, du bruit, mode de sortie, période) 141
- combinaison des canaux de sons purs
- combinaison de bruits
- mode de sortie
- la durée du son

ANNEXES 144

A) Récapitulation des fonctions et commandes du basic 145

B) Les messages d'erreur 171

C) Utilisation d'un magnétophone à cassettes 178

D) Utilisation d'une imprimante 181

E) Numérotation hexadécimale - représentation des nombres 183

F) Tableau des codes ASCII 187

G) Carte mémoire de l'Oric 188

H) Rôle des attributs graphiques 189

CHAPITRE **1**

Présentation générale de l'oric

1.1 INTRODUCTION

L'Oric est un microordinateur : il est extrêmement petit par rapport aux premiers ordinateurs et son "coeur" est un microprocesseur. Ainsi que vous pouvez le voir sur la photo A, l'Oric a un clavier utilisé pour "entrer" des données et de nombreuses prises pour relier la télévision et les extensions. A l'intérieur sont placés les circuits intégrés parmi lesquels figurent le microprocesseur, les mémoires... (voir paragraphe 1.3). Au début, vous n'avez besoin d'aucune connaissance en électronique pour utiliser votre Oric.

La sortie vidéo est présente à l'arrière de l'appareil et délivre des signaux RVB que l'on transmet à la télévision par l'intermédiaire de la prise Péritel, fournie avec l'Oric (voir paragraphe 1.4) et qui est installée obligatoirement sur tous les téléviseurs en couleurs produits depuis 1980, en France.

La photo A montre un Oric 1 avec un gros plan sur le clavier.

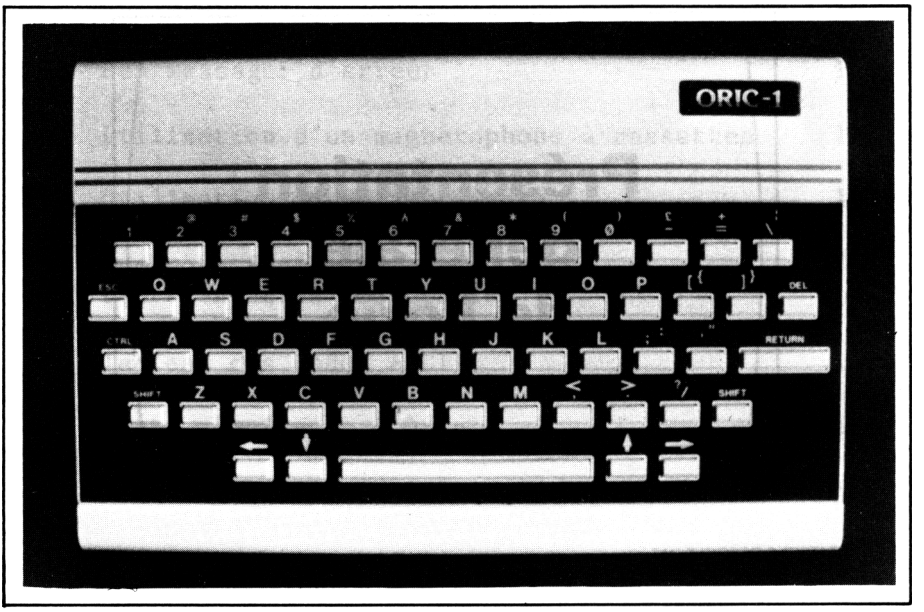


PHOTO A : LE SYSTEME ORIC

Votre microordinateur Oric offre de nombreuses possibilités graphiques et sonores. Avec un affichage graphique haute-résolution de 200 x 240 points sur l'écran, plusieurs couleurs, vous pourrez utiliser des jeux graphiques, tracer des courbes sur l'écran.... Avec la sortie sonore en connexion avec une chaîne hi-fi, vous pourrez composer de la musique...

Avec l'Oric, vous pouvez vous contenter de programmes tout faits (jeux, gestion familiale...), mais vous pouvez aussi composer vos programmes, utiliser des cassettes, communiquer avec d'autres Oric.... Votre microordinateur s'avérera bientôt indispensable.

1.2 LE CLAVIER DE L'ORIC

Le clavier incorporé sur la face avant du boîtier comporte cinquante-sept touches. Il permettra de générer

les lettres majuscules et minuscules, les touches fonctions utilitaires (ESC, RETURN, DEL...), les caractères spéciaux (ponctuation, opérations...). Il comporte en outre une barre centrale permettant de générer des blancs, quatre touches situées de part et d'autre de la barre centrale et permettant de se déplacer sur l'écran (gauche/droite/bas/haut). Toutes les touches sont à répétition automatique, c'est-à-dire qu'il suffit de laisser le doigt appuyé sur la touche pour répéter le caractère plusieurs fois.

Les touches du clavier de l'Oric sont tactiles. Vous sentirez avec votre doigt qu'une touche sera enfoncée, et vous entendrez de plus un léger bruit caractéristique. Cela permet par exemple de compter à l'oreille le nombre de caractères, sans regarder le clavier.

L'Oric ayant été créé en Angleterre, la disposition des touches sur le clavier est identique à celle des machines à écrire anglaises (QWERTY). Toutefois, ceci ne doit pas vous gêner si vous connaissez les machines à écrire françaises, la différence étant limitée aux lettres A, Z, Q, W, et M.

Le clavier de l'Oric est simple d'emploi et vous serez vite familiarisé avec lui.

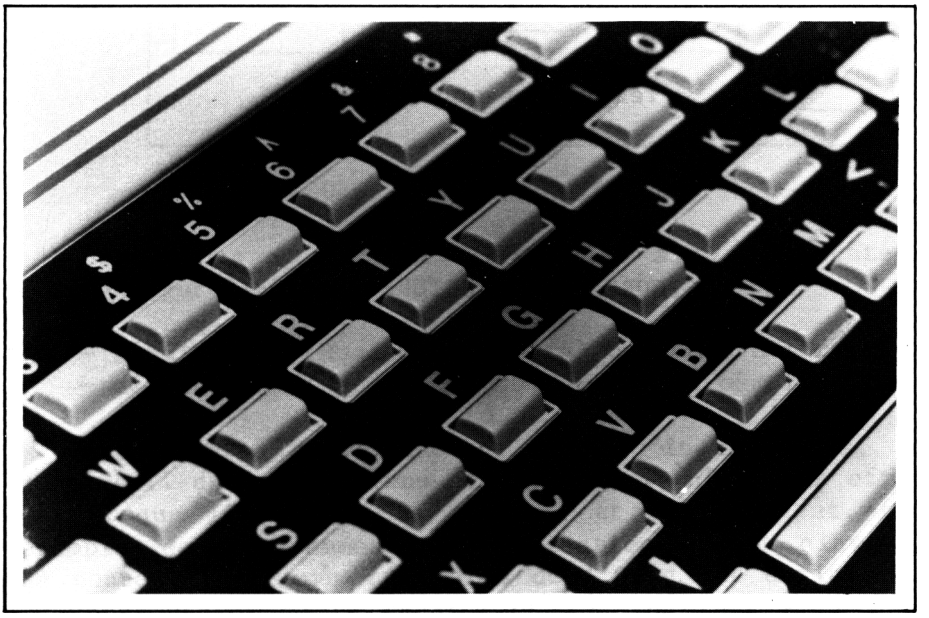


PHOTO B : VUE DU CLAVIER

1.3 UNITE CENTRALE

L'Oric est avant tout une machine à traiter l'information. Elle reçoit, stocke, traite et communique des données avec le clavier, un téléviseur couleur, ou avec un magnétophone à cassettes.

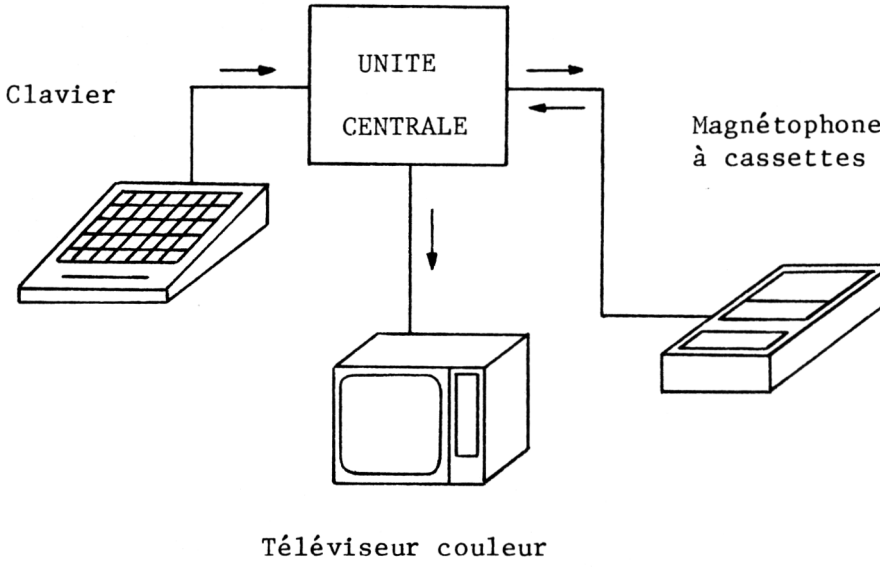


FIGURE 1 : STRUCTURE DE L'ORIC

Le coeur de votre microordinateur est appelé unité centrale (en anglais **Central Processor Unit**, ou **CPU** en abrégé). Cette unité centrale supervise le fonctionnement du système Oric tout entier.

Elle décide quand aller chercher les données au clavier ou en envoyer sur l'écran, sur la cassette... Elle exécute les programmes que **vous** lui demandez.

L'unité centrale est organisée autour d'un simple circuit intégré (boîtier noir muni de pattes) appelé microprocesseur. Ce boîtier fait les calculs, va chercher les données et les range. Pour pouvoir travailler assez rapidement, le microprocesseur ne va pas demander une à une les données au clavier ; il a besoin pour cela d'autres circuits, appelés mémoires, qui contiennent le programme et la plupart des données utiles au programme.

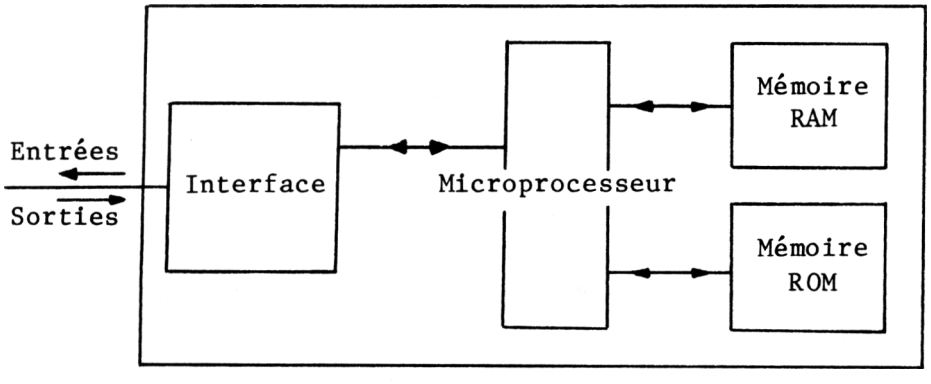


FIGURE 2 : UNITE CENTRALE

Deux types de mémoire existent et sont représentées sur la figure 2 :

- mémoire ROM (Read Only Memory) ou mémoire morte ;
- mémoire RAM (Random Access Memory) ou mémoire vive.

Une mémoire morte est un stock d'instructions, de données... dont le contenu est permanent, n'est pas effacé lorsque l'on coupe le courant, et qui contient, par exemple, des programmes de communication avec le clavier. Le contenu d'une mémoire morte de l'Oric est défini par le constructeur de la machine et vous ne pourrez y toucher.

Dans une mémoire RAM (ou vive), on peut écrire, lire ou effacer de l'information aussi souvent qu'on le souhaite ; elle contiendra, par exemple, les programmes que vous taperez, mais elle sera effacée par toute coupure du courant.

Le microprocesseur de l'Oric est le même que celui de l'Apple II ou du Vic 20. La taille des mémoires de l'Oric peut varier de 16 000 à 48 000 caractères en mémoire vive.

La mémoire morte contient l'équivalent de 16 000 caractères pour stocker le langage de programmation Basic auquel nous vous initierons aux chapitres 2 et 3.

Dans la suite de ce livre, nous utiliserons le symbole K pour représenter le chiffre 1024 (= 2^{10}).

Ainsi, par exemple, 16 k = 16.384 soit 16.000 environ.

La photo C, ci-dessous, montre l'intérieur de votre Oric. Le microprocesseur, les mémoires RAM, les mémoires ROM ainsi que le haut-parleur y sont clairement visibles.

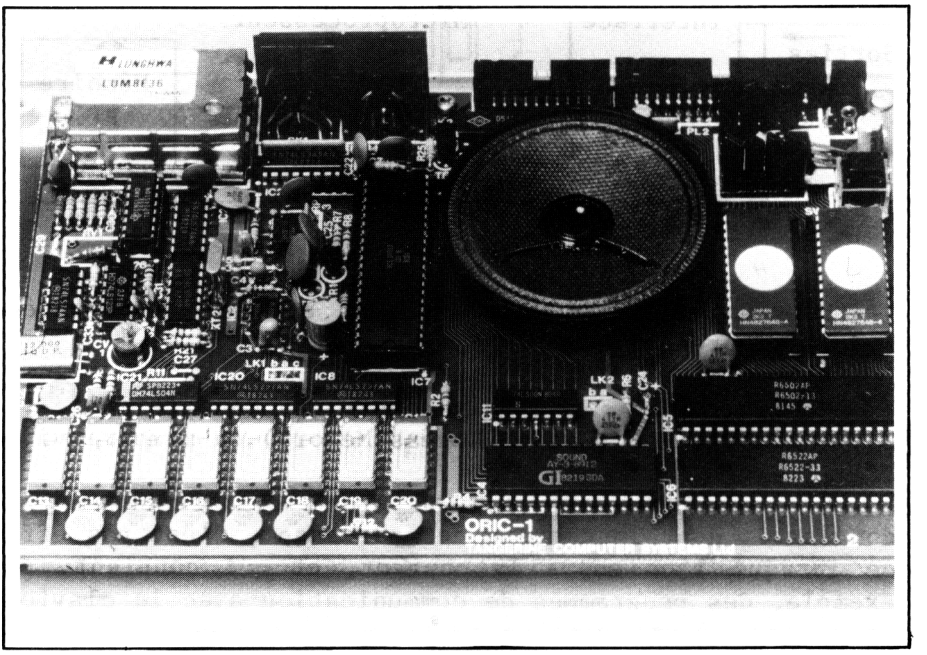
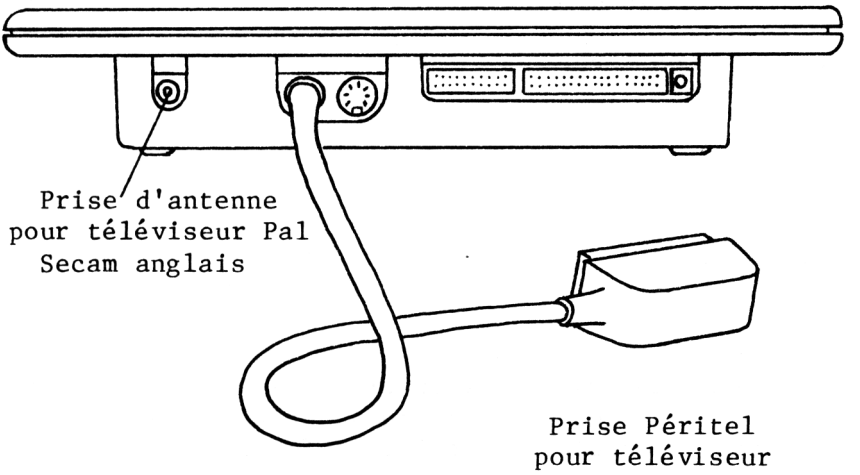


PHOTO C : STRUCTURE INTERNE DE L'ORIC

1.4 SORTIE VIDEO

Deux prises existent sur la face arrière de l'Oric pour le relier à un téléviseur couleur :



Vous disposez donc de deux options :

a. branchement sur l'entrée Péritel pour téléviseur couleur
b. branchement sur l'entrée Prise d'antenne pour téléviseur anglais.

La première solution vous donnera une qualité nettement meilleure que la seconde (couleurs, netteté...).

Si vous n'avez pas de prise Péritel sur votre téléviseur, il est tout à fait possible de la faire installer par un revendeur spécialisé.

1.5 SORTIE CASSETTE

La connection se fait par l'intermédiaire d'une prise DIN. C'est le format classique Tangerine qui est utilisé.

Les vitesses d'enregistrement vont de 40 à 300 caractères par seconde. Un circuit spécial de l'Oric élimine les bruits parasites. Tous les types d'informations peuvent être sauvegardés (programmes, données, blocs mémoire...).

Il serait prématuré d'énumérer ici les commandes spécifiques à l'interface cassette, attendu que vous n'avez encore aucun programme ou exécution de programme à sauvegarder sur cassette. Une annexe sera consacrée à ces commandes (de même pour la sortie imprimante), et vous vous y reporterez. Pour l'heure, nous nous contenterons d'une description technique.

1.6 EXTENSIONS

Reportez-vous à la figure 1 de la page suivante. Vous pouvez y voir, à côté du branchement imprimante, un branchement pour extensions. Ces possibilités d'extension sont des connexions de cartouches ROM par exemple, c'est-à-dire que vous pouvez brancher une ligne pour étendre les mémoires. Vous pouvez aussi connecter des cartouches de jeux, un lecteur de disques, et bien sûr un modem (modulateur-démodulateur) qui transformera les signaux codés constituant votre programme pour les rendre aptes au transport par ligne téléphonique. C'est le système Prestel Mailbox qui est utilisé.

1.7 MISE EN ROUTE

Oric n'est déjà plus un inconnu, mais il reste à le mettre en route.

Vous vous trouvez devant :

- l'ordinateur, avec son clavier ;
- deux adaptateurs
  . le plus gros sert à l'alimentation,
  . le plus petit fournit un voltage indispensable pour permuter la prise Péritel en mode vidéo. L'alimentation Péritel permet de sélectionner l'entrée RVB parmi les trois entrées possible.
- une prise Péritel. Selon l'importateur d'Oric, la prise Péritel peut intégrer le petit adaptateur précédent.

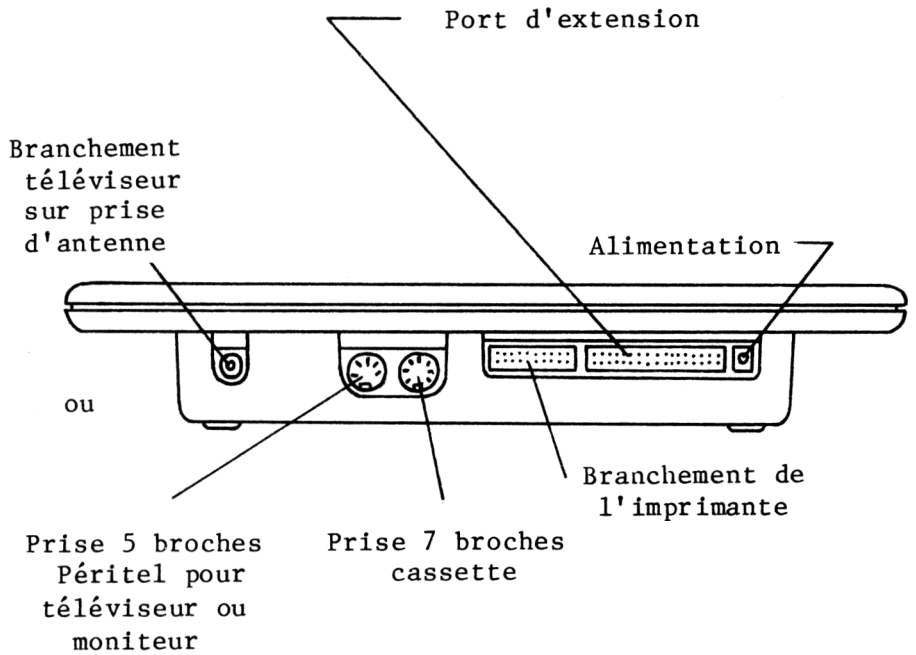


FIGURE 1

Sur l'extrémité de la prise Péritel correspondant au récepteur, vous enfichez la prise en croix reliée au plus petit des deux adaptateurs.

Oric n'a pas de touche ARRET/MARCHE. Il faut couper l'alimentation pour l'arrêter.

Vos branchements faits, vous devez voir apparaître sur l'écran :

```
ORIC EXTENDED BASIC V1.0
(C) 1983 TANGERINE
... BYTES FREE
Ready
```

FIGURE 2

Voilà, vous êtes prêt à découvrir le monde de l'informatique.

Que vos goûts vous conduisent vers les mathématiques, la gestion, le dessin, la musique ou l'informatique en elle-même, vous allez devenir plus efficace.

CHAPITRE **2**

Initiation à la programmation

2.1 INTRODUCTION

Si vous avez suivi les instructions du chapitre concernant la mise en route, vous savez maintenant effectuer rapidement les branchements nécessaires.

Voici donc l'appareil sous tension, convenablement relié à votre récepteur de télévision. Si vous n'avez pas fait d'erreur de branchement, vous voyez apparaître sur l'écran, en noir sur fond blanc :

```
ORIC EXTENDED BASIC V1.0
(C) 1983 TANGERINE
47870(*) BYTES FREE
Ready
```

Si tel n'était pas le cas, reportez-vous au chapitre 1.

Le rectangle noir clignotant, qui apparaît sur le bord supérieur gauche de l'écran, sous le mot Ready, s'appelle le curseur. Il détermine l'emplacement du caractère que l'on veut écrire, et se déplace au fur et à mesure sur la ligne.

(*) Ce nombre peut être inférieur, selon la version d'Oric en présence.

Essayons par exemple de taper au clavier :

BONJOUR

Le mot BONJOUR s'écrit sur l'écran au fur et à mesure que nous le tapons. Oric va-t-il nous répondre ?

Nous avons terminé, et... il ne se passe apparemment rien. En fait, pour qu'Oric prenne en compte une ligne que l'on vient de taper, il faut le lui demander en appuyant sur la touche marquée [RETURN].

Oric répond alors :

? SYNTAX ERROR

Ready

Que s'est-il passé ? L'ordinateur a lu notre message, a constaté qu'il n'était pas écrit suivant la "bonne" syntaxe et nous a signalé le fait par un message d'erreur.

Tapons maintenant :

PRINT "BONJOUR"

Pour taper les guillemets il faut enfoncer l'une des touches [SHIFT], la maintenir, et appuyer en même temps sur la touche marquée [,"]. De manière générale, il faut procéder de la même façon pour obtenir le caractère du haut sur les touches à double fonction.

Appuyons ensuite sur [RETURN].

En principe, tout message que vous taperez devra être suivi de la touche [RETURN] ; nous vous le rappellerons d'abord systématiquement puis, peu à peu, nous l'omettrons. Elle sera sous-entendue mais devra toujours être tapée.

Si vous avez composé exactement les lettres ci-dessus, sans oublier les guillemets, vous obtenez immédiatement sur l'écran :

BONJOUR

Ready

Oric a imprimé BONJOUR et nous informe, en affichant Ready, qu'il est prêt à enregistrer d'autres messages.

Nous voyons donc que pour communiquer avec Oric, il faudra toujours taper nos messages en respectant rigoureusement une certaine syntaxe, qui fonde en fait un véritable langage.

Avant de voir plus précisément ce qu'est ce langage, il faut introduire un certain nombre de notions :

2.1.1 Stockage de l'information - Bit - Octet

Oric est un microordinateur, dérivé des grands ordinateurs qui fonctionnent actuellement dans le monde industriel. Mais ne nous y trompons pas : s'il a une taille très réduite, son principe de fonctionnement est en tout point semblable à celui de ses "aînés". Comme eux, c'est une machine de traitement automatique de données.

Et une des opérations essentielles qu'Oric - comme tout ordinateur - doit pouvoir effectuer sur l'information, est sa mémorisation.

A un autre niveau, c'est ce qui se passe lorsque vous mettez un nombre en mémoire dans votre calculatrice de poche.

La meilleure méthode que l'on connaisse actuellement pour mémoriser une information est un codage binaire, c'est-à-dire qui réduit tout élément à **deux** états bien distincts. Ce peut être par exemple l'absence **ou** la présence de trou dans une carte perforée, un élément aimanté dans un sens **ou** dans l'autre.

Oric, lui, utilise des éléments de circuits électroniques qui peuvent être portés à une tension de 5V, ou bien rester à 0.

L'information contenue dans un tel élément s'appelle **BIT** (abréviation de l'expression anglaise **BINARY DIGIT**, qui signifie chiffre binaire) et contient en quelque sorte la réponse par oui ou par non à une certaine question.

Mais un seul **BIT** contient une information trop "petite" pour être manipulée aisément. C'est pourquoi on les regroupe généralement par paquets de 8, ou **OCTETS.**

Il y a une remarque intéressante à faire ici. On se souvient qu'Oric, dès qu'on le met sous tension, affiche:

47870 BYTES FREE

Or en anglais **OCTET** se dit **BYTE.** Si l'on sait que **FREE** signifie **LIBRE,** on voit que la mémoire d'Oric a une capacité de 47870 octets.

En pratique, sachez que sur un octet il est possible de représenter les nombres inférieurs à 256 ou les caractères du clavier. Les autres nombres seront représentés sur plusieurs octets.

Les lecteurs intéressés par ce codage peuvent lire la suite de ce paragraphe. Les autres personnes passeront au paragraphe 2.1.2.

Un octet étant un paquet de 8 bits, supposons que l'on représente les deux états que peut prendre un bit par 0 et 1. Un octet pourra alors s'écrire comme une succession de huit 0 ou 1 ; par exemple 00100010, ou 11101001... En fait, comme il y a deux possibilités pour chaque bit, on a 256 possibilités différentes pour un octet.

On peut maintenant comprendre comment un **OCTET** pourra représenter un nombre : il suffit de considérer que la suite de huit 0 ou 1 qui constitue un octet est un nombre écrit dans le système de numérotation binaire.

Ainsi 00100010 représentera :

$0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 32 + 2 = 34.$

Cela n'a absolument rien de mystérieux. Nous faisons chaque jour la même chose lorsque nous écrivons un nombre en base dix. Par exemple 534 vaut $5 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$.

Mais avec notre convention, on s'aperçoit que le nombre le plus grand qu'on puisse écrire est 11111111 = 255. Pour pouvoir représenter des nombres plus grands que 255, il faudra qu'un nombre occupe plus d'un octet, mais le principe reste le même.

Stocker des nombres est donc finalement relativement facile. Mais comment faire pour stocker des lettres ou des caractères ? Là encore, rien de plus simple. Il suffit de considérer par exemple les codes ASCII où A = 01000001, B = 01000010, et ainsi de suite pour toutes les lettres. Comme il y a 256 possibilités dans un octet, on peut avoir un jeu de 256 caractères différents, ce qui permet de représenter toutes les lettres, minuscules et majuscules, les chiffres, les caractères de ponctuation, etc.

2.1.2 Notion de langage - Le Basic

Comme on peut le penser d'après la façon de stocker l'information, le seul langage qu'Oric "comprenne" vraiment est le langage binaire, composé d'une succession de **BITS**, et qu'on appelle encore "langage machine".

Cela signifie-t-il que nous sommes obligés d'introduire toutes nos commandes et instructions sous la forme d'une succession de 0 et de 1 ? Heureusement non ! Ce serait beaucoup trop compliqué à écrire et à comprendre (du moins pour nous, mais pas pour la machine, qui est littéralement infatigable).

Pour nous éviter cette peine, il existe ce que l'on appelle des langages évolués. Ce sont d'autres langages de programmation que le langage machine, mais qui sont beaucoup plus proches du langage humain et des notations mathématiques usuelles. Par suite, ils sont beaucoup plus aisés d'emploi.

Il existe un certain nombre de tels langages, chacun étant plus ou moins adapté à un usage spécifique. Parmi les plus connus, on peut citer FORTRAN, COBOL, PASCAL, PL1, ... et BASIC, qui est le langage employé par Oric.

BASIC est le langage actuellement le plus répandu sur

les microordinateurs. Sa très grande diffusion tient à sa facilité d'apprentissage : BASIC est en effet le langage le plus facile à apprendre et à utiliser par les débutants.

Il a été créé en 1964 pour permettre l'apprentissage facile de la programmation, et son nom est l'abréviation de **Beginner's All purpose Symbolic Instruction Code**, ce qui signifie : codage symbolique des instructions d'usage général pour les débutants.

Nous savons donc qu'Oric "comprend" le BASIC ; il nous reste maintenant à apprendre nous-même ce langage, pour pouvoir utiliser notre microordinateur. C'est le but des chapitres qui suivent. Nous verrons que ce n'est pas très difficile, car BASIC est un langage bien plus facile à apprendre qu'une langue étrangère : il ne comprend en effet qu'une centaine de mots !

Remarquons que BASIC est susceptible de varier légèrement d'un microordinateur à l'autre, un peu de la même façon, par exemple, qu'une langue est susceptible d'avoir des variantes locales d'une région à l'autre.

Ce n'est pas grave dans la mesure où l'on programme toujours sur le même microordinateur. Mais on peut être amené à changer de machine. (Attention : tous les Orics "parlent" bien le même BASIC ; les différences sont sensibles entre des machines de constructeurs différents). Aussi ce chapitre présente-t-il les instructions de base communes à toutes les versions de BASIC. Les instructions plus élaborées, ainsi que les instructions BASIC spécifiques à Oric sont présentées dans le chapitre suivant.

2.2 MODE DIRECT - MODE DIFFERE - NOTION DE PROGRAMME

2.2.1 Introduction

Nous avons déjà rencontré une instruction BASIC, c'est l'instruction PRINT. Comme son nom l'indique (en anglais, PRINT signifie imprimer) elle permet d'écrire sur l'écran. Son emploi est très simple. De la même façon que nous avons su faire écrire BONJOUR, nous pourrons écrire une ligne de texte quelconque en tapant cette ligne après PRINT, sans oublier de mettre des guillemets (") de part et d'autre de la ligne de texte, comme dans l'exemple suivant :

PRINT "CECI EST UNE LIGNE DE TEXTE"

Tapons maintenant :

PRINT 2 + 3

(Pour taper + il faut appuyer simultanément sur [SHIFT] et sur la touche [+=]).

Appuyons sur [RETURN]

Il s'affiche alors immédiatement sur l'écran :

5

Ready

Oric a effectué le calcul et a immédiatement imprimé le résultat.

Ce mode de fonctionnement, dans lequel une instruction est exécutée immédiatement après avoir été tapée, est dit "mode immédiat", ou "mode d'exécution immédiate", ou encore "mode direct".

C'est très proche du fonctionnement de votre calculatrice de poche où les calculs se font au fur et à mesure que vous les rentrez. La seul différence, c'est qu'il faut d'abord taper PRINT.

Nous y reviendrons. Avant cela, il faut voir le second mode de fonctionnement.

Tapons :

20 PRINT 2 + 3

Appuyons sur [RETURN] ... cette fois, il ne se passe rien !

Tapons encore :

10 PRINT "BONJOUR"

Appuyons sur [RETURN] : toujours rien. Serait-ce une panne ?

Mais tapons maintenant :

```
RUN
```

et appuyons sur [RETURN]. Il apparaît alors à l'écran :

```
BONJOUR
5

Ready
```

Expliquons ce qui s'est passé. Oric, au lieu d'exécuter immédiatement les instructions, les a mises en mémoire où elles constituent ce que l'on appelle un PROGRAMME.

En tapant RUN, nous avons signalé à Oric qu'il pouvait exécuter ce programme, c'est-à-dire exécuter les instructions les unes après les autres. On dit qu'il a fonctionné suivant le "mode différé", ou "mode programmé", son second mode de fonctionnement.

Ce mode se caractérise par la présence d'un numéro en tête de chaque ligne. C'est ce qui permet de déterminer le mode dans lequel on se trouve, car, avec quelques exceptions, toutes les instructions utilisables en mode programmé le sont aussi en mode direct.

Le numéro en tête de ligne a aussi une autre utilité. Il fixe l'ordre dans lequel seront exécutées les instructions. Nous avons pu remarquer en effet que la ligne 10 a été exécutée avant la ligne 20, bien qu'elle ait été tapée après. En mode programmé, l'ordre dans lequel on tape les lignes n'a donc aucune importance ; Oric exécutera de toute façon les instructions par ordre de numéros de ligne croissants.

Nous verrons bien vite que le mode de fonctionnement le plus intéressant d'Oric est le mode programmé.
Néanmoins, pour mieux nous familiariser avec son fonctionnement, nous allons nous intéresser d'abord au mode direct.

2.2.2 Fonctionnement en mode direct

Nous allons, en fait, apprendre d'abord à nous servir d'Oric comme d'une calculatrice de poche, en apprenant à évaluer les expressions arithmétiques.

- Les opérations élémentaires

. **ADDITION**

Nous savons déjà additionner : pour ajouter 12 à 27, nous avons compris qu'il suffit de taper :

```
PRINT 12 + 27
```

N'oublions pas d'appuyer sur [RETURN] (ce sera désormais sous-entendu). Oric affiche aussitôt le résultat :

```
39

Ready
```

Rappelons que **READY** (mot anglais pour PRET) signifie qu'Oric a terminé d'exécuter les instructions que nous lui avions fournies et qu'il se tient prêt à enregistrer de nouvelles commandes.

. **SOUSTRACTION**

La soustraction ne fait pas plus de difficulté ; essayons par exemple :

```
PRINT 8 - 2.5
```

On obtient 5.5. Notons le point à la place de la virgule. Oric, comme tous les ordinateurs, utilise la notation arithmétique anglo-saxonne, dans laquelle on écrit 12.25 au lieu de 12,25.

Supposons qu'à la ligne précédente, nous nous soyons trompés et ayons tapé :

```
PRINT 8 - 2,5
```

Nous obtenons alors :

```
6       5
```

Oric a considéré la virgule comme un séparateur entre deux valeurs. Il comprend qu'il faut imprimer 8 - 2, soit 6, puis la valeur 5. Ce n'est pas du tout ce que nous voulions obtenir, et cela nous montre combien il faut être précis dans ce que nous tapons :
Oric obéit **à la lettre** à ce que nous lui commandons.

Néanmoins, malgré tout le soin que nous apportons en tapant nos messages, une erreur est toujours possible. Comment faire pour la corriger ? Reprenons cet exemple :

PRINT 8 - 2,5

mais nous n'avons pas encore appuyé sur la touche [RETURN].

Pour corriger notre erreur, et remplacer la virgule par un point, il faut utiliser la touche [DEL] : elle permet de faire reculer le curseur.

Dans notre exemple, appuyons une fois sur [DEL] : le curseur recule d'un cran et 5 disparaît. Appuyons une autre fois, et la virgule disparaît. Nous pouvons alors taper un point et retaper le 5 : notre erreur est réparée.

ATTENTION : Il ne faut pas utiliser la touche [←] ; le curseur recule bien aussi, comme avec la touche [DEL] ; on pourra même remplacer la virgule par un point, et cela sans que le 5 s'efface ; mais en fait ces modifications n'auront eu lieu que sur l'écran : Oric ne les aura pas prises en compte.

L'usage de la touche [←] et des touches analogues sera abordé ultérieurement.

Si nous avons fait beaucoup d'erreurs dans la même ligne et que nous préférons la retaper entièrement, nous pouvons appuyer simultanément sur [CTRL] et sur [X]. Le caractère \ apparaît au bout de la ligne, dont il ne sera plus tenu compte : bien qu'elle soit toujours imprimée sur l'écran, Oric considère la ligne comme effacée.

Nous pouvons alors la retaper car, simultanément avec l'apparition du caractère \ , le curseur a sauté au début de la ligne suivante.

. **MULTIPLICATION**

Le symbole de la multiplication est * (qu'on obtient en appuyant simultanément sur les touches [SHIFT] et [8]).

PRINT 4 * 16

donne 64

Peut-être trouvez-vous un peu fastidieux de taper les cinq lettres du mot PRINT avant chaque opération ? En fait, il suffit de taper [?] au lieu de PRINT. C'en est en quelque sorte une abréviation qui accélère la frappe. Oric comprend les deux termes de la même façon. Dans la suite, nous utiliserons indifféremment ? ou PRINT.

. **DIVISION**

Pour effectuer une division, le symbole est / qui se prononce SLASH.

Essayons :

```
? 2/3
```

N'oublions pas d'appuyer sur [RETURN] !

On obtient :

```
.666666667
```

Le zéro avant le point décimal est omis, mais on pourra remarquer qu'il est barré pour ne pas le confondre avec la lettre O.

. **PUISSANCE**

Il reste une dernière opération élémentaire, très utile, l'élévation à la puissance. Son symbole est ↑ (que l'on obtient par [SHIFT] [6]).

```
PRINT 4 ↑ 3
```

donne 64 (= 4^3).

On peut ainsi obtenir par exemple la racine carrée par :

```
? 25 ↑ 0.5
```

qui donne 5

Signalons qu'il existe une autre possibilité : la fonction SQR. On l'utilise de la façon suivante :

```
? SQR (25)
```

Elle donne aussi la racine carrée, mais le calcul en est plus rapide.

- Les ordres de priorité des opérateurs

Nous avons donc découvert les 5 opérations mathématiques fondamentales :

+, -, *, /, ↑

Mais nous n'avons vu que des exemples très simples. On peut calculer des expressions plus compliquées contenant plusieurs opérateurs. Essayons les quelques exemples suivants :

```
? 5 + 3 * 4
? 8 - 6 / 2
? 5 * 8 / 4
? 2 * 3 ↑ 2
```

Le premier a pour résultat 17 (= 5 + 12) et non pas 32 (= 8 * 4) car Oric évalue d'abord la multiplication. On dit que la multiplication est prioritaire sur l'addition.

De même, la deuxième expression donne 5 : on évalue d'abord la division.

Pour la troisième, on obtient 10 : les calculs sont effectués dans l'ordre où ils se présentent, car * et / ont même priorité.

Enfin le quatrième exemple a pour résultat 18 : l'élévation à la puissance est effectuée en premier car elle est prioritaire sur la multiplication.

Pour nous résumer, il faut retenir que les calculs sont effectués de gauche à droite, sauf s'il y a des priorités à respecter entre les divers opérateurs. Ces priorités sont les suivantes :

↑	: est toujours effectué en premier, puis
*, /	: qui ont la même priorité, et enfin
+, -	: qui sont aussi "ex aequo".

Pour changer l'ordre de priorité, on emploie des parenthèses, car une expression entre parenthèses est toujours évaluée la première. Par exemple :

(4 + 3) * 2 donne 14 alors que 4 + 3 * 2 donne 10
(2 * 3) ↑ 2 donne 36 alors que 2 * 3 ↑ 2 donne 18
7 - (3 + 2) donne 2 alors que 7 - 3 + 2 donne 6

Il peut y avoir des parenthèses emboîtées, mais alors il faut veiller à ce qu'il y ait bien autant de parenthèses ouvrantes "(" que de parenthèses fermantes ")", pour que l'expression ait un sens.

Les parenthèses emboîtées suivent la règle de la plus intérieure vers la plus extérieure, et de la gauche vers la droite.

Ainsi,

(4 + 2) * 3 ↑ (7 - (3 + 2)) est correct.

Mais ((8 - 6) * ((3 - 2) ↑ 6) / 2

provoquera un :

? SYNTAX ERROR

- Notion de variable

Au stade où nous en sommes, nous savons évaluer des expressions, même très compliquées, mais tout ce que nous pouvons en faire, c'est faire imprimer leur valeur sur l'écran.

En fait, il est possible de mettre ces valeurs en mémoire pour les utiliser ultérieurement. Comment faire ? Eh bien, il suffit de donner un nom à l'expression que l'on veut mettre en mémoire, comme sur cet exemple :

B = 2 + 3

Même après avoir appuyé sur [RETURN], il n'apparaît rien d'autre que Ready sur l'écran. Oric a calculé l'expression et lui a attribué automatiquement un emplacement en mémoire qu'il a nommé B.

On dit que l'on a constitué une variable de nom B.

On peut maintenant utiliser cette variable dans d'autres expressions ; tapons :

```
PRINT B
```

nous obtenons :

```
5
```

Tapons :

```
PRINT 3 * B
```

nous obtenons :

```
15
```

Nous pourrions imaginer ainsi un grand nombre d'expressions, utilisant la variable B, mais sa valeur ne sera jamais détruite (c'est-à-dire qu'elle ne sera jamais effacée de la mémoire) sauf lors d'une nouvelle instruction d'affectation ; si nous tapons :

```
B = 3
```

alors `PRINT B + 1`

donne `4`

REMARQUE : Au lieu de taper B = 3 nous aurions pu écrire LET B = 3 (ce qui signifie "affecter à B la valeur 3"). Ces deux formes sont absolument équivalentes, Oric ne fera pas de différence. En fait, il est plus simple de taper B = 3 ; si nous signalons l'instruction LET c'est que, avec certains BASIC implantés sur d'autres microordinateurs, cette instruction est obligatoire, ce qui n'est pas le cas pour Oric.

Nous avons considéré la variable B, mais nous aurions pu aussi l'appeler C, ou A, ou NOMBRE.... En fait les noms des variables sont pratiquement arbitraires. Il suffit de respecter les 3 règles suivantes :

- Le premier caractère doit être une lettre : A, B ... Z
- Les caractères suivants peuvent être des lettres ou des chiffres (Ex : B2, NOMBRE, K12J4...).

- Le nom de la variable ne doit pas contenir un des mots particuliers au langage BASIC, qu'on appelle MOTS-CLES. Appeler une variable LETTRE est interdit, car il y a LET, mot-clé que nous venons de voir. De même CHIFFRE n'est pas utilisable, car il contient IF, mot-clé que nous verrons par la suite.

  La liste complète de ces mots-clés est donnée en annexe A.

Une précision importante à retenir, car elle peut être la source de nombreuses erreurs, est la suivante :

Quel que soit le nombre de caractères que comporte le nom d'une variable, Oric ne prend en compte que les deux premiers.

Des variables distinctes comme NOMBRE et NOM sont considérées comme identiques, car les 2 premières lettres sont semblables.

Alors, si nous tapons :

```
NOMBRE = 2
NOM = 10
```

et si ensuite nous faisons :

```
PRINT NOMBRE
```

nous obtenons :

```
10
```

et non pas 2. En faisant NOM = 10, Oric a remplacé la valeur 2 qu'avait la variable "NO" (seules lettres qu'a "vues" Oric) par la valeur 10. En faisant écrire NOMBRE (c'est-à-dire pour Oric, toujours la variable "NO") nous avons naturellement obtenu la dernière valeur entrée.

Nous sommes maintenant armés pour aborder le mode programmé, ce qui devrait se révéler plus intéressant que ce qui précède, peut-être un peu rébarbatif, mais absolument nécessaire.

2.3 LES BASES DE LA PROGRAMMATION - DECOUVERTE DU LANGAGE BASIC

2.3.1 Le premier programme

Nous savons déjà ce qu'est un programme ; nous avons vu que c'est un ensemble d'instructions, dont chaque ligne comporte un numéro en tête, et qui sont mises en mémoire.

On lance l'exécution en tapant RUN ; les instructions sont alors exécutées les unes après les autres, par ordre de numéros de ligne croissants.

Notre premier programme va être très simple. Il a simplement pour but de calculer la circonférence C d'un cercle de diamètre D ($C = \pi D$).

```
10 INPUT D
20 C = PI * D
30 PRINT C
```

Malgré sa simplicité, ce petit programme comporte les étapes fondamentales de tout traitement d'information :

. L'ACQUISITION DES DONNEES à la ligne 10. Nous y découvrons un nouveau mot de BASIC : INPUT. Son effet est le suivant : lorsque nous lancerons le programme en tapant RUN, Oric affichera un ?, suivi du curseur clignotant. Il attendra ensuite que nous tapions un nombre (qui sera le diamètre du cercle). Supposons que nous tapions la valeur 2 (suivie comme toujours de [RETURN]). Oric affectera cette valeur à D, puis passera à la ligne suivante où sera effectué :

. LE TRAITEMENT DES DONNEES à la ligne 20. Il est évidemment très réduit ici ; on effectue simplement la multiplication de D (qui vaut maintenant 2) par PI ($= \pi$). Cette dernière valeur est "connue" par Oric. Nous pourrons nous en assurer en tapant ultérieurement, en mode direct :

```
PRINT PI
```

et nous obtiendrons :

```
3.14159265
```

Après le traitement des données, c'est-à-dire le calcul des résultats, vient la dernière étape :

. LA SORTIE DES RESULTATS à la ligne 30 où l'on affiche la valeur de la circonférence (ici 6.2831853) grâce à l'instruction PRINT, que nous connaissons bien maintenant.

Si vous avez bien suivi ces trois étapes du fonctionnement d'un programme, le reste ne posera pas de problème, car pratiquement toute la méthode de la programmation est contenue dans ces quelques lignes.

Notre petit programme est susceptible d'être amélioré, bien qu'il fonctionne parfaitement : pour connaître la circonférence d'un cercle de diamètre donné il suffit de taper RUN, puis, après le point d'interrogation qui apparaît alors, de taper le diamètre du cercle ; la circonférence s'affiche immédiatement après qu'on a enfoncé la touche [RETURN].

Si l'on veut la circonférence d'un cercle de diamètre différent, il suffit de retaper RUN et le programme est réexécuté. Et il le sera autant de fois qu'on retapera RUN : cela n'usera pas le programme de le faire exécuter, il restera toujours en mémoire (sauf si on débranche Oric)!

Quelles sont les améliorations que l'on aimerait apporter ? Tout d'abord, il y a un problème à l'entrée des données. En effet, l'instruction INPUT provoque l'affichage d'un ? seul. Nous savons qu'il faut alors taper la valeur du diamètre du cercle, mais nous le savons uniquement parce que nous avons écrit le programme... Autrement, rien ne nous l'indique, et une personne qui ne connaîtrait pas le programme ne saurait pas quoi taper après le ?.

Il faudrait qu'Oric indique, par un message avant le ?, que c'est le diamètre du cercle qu'il faut taper et pas autre chose.

Cela peut sembler inutile ici, car le programme est simple, mais imaginons un très gros programme qui comporterait de nombreux INPUT. Même si nous l'avions nous-même écrit, il ne serait pas inutile qu'il nous rappelle dans quel ordre entrer les données.

Cela s'obtient très facilement, car l'instruction INPUT peut aussi s'employer sous la forme suivante :

INPUT "texte" ; variable

Le texte entre guillemets apparaît alors, suivi d'un point d'interrogation. Il suffit ensuite de taper la valeur demandée, qui est affectée à la variable.

Dans notre programme, nous aurons résolu notre problème en remplaçant la ligne 10 par :

10 INPUT "QUEL EST LE DIAMETRE DU CERCLE" ; D

La syntaxe doit être scrupuleusement observée ; en particulier les guillemets de part et d'autre du texte, ainsi que le point-virgule entre ce dernier et la variable sont absolument obligatoires.

Mais nous n'avons résolu un problème que pour en faire surgir un autre. Car comment remplacer la ligne 10 dans notre programme ? En fait, c'est tout simple, il suffit de taper la nouvelle ligne 10, sans oublier le numéro. L'ancienne ligne 10 sera alors effacée de la mémoire d'Oric où se trouve le programme, et remplacée par la nouvelle ligne.

Cette remarque est très générale : pour remplacer une ligne dans un programme, il suffit simplement de la retaper, sans oublier son numéro au début.

Le problème à l'entrée des données est donc résolu, mais la situation est analogue lors de l'impression des résultats : on obtient un nombre, mais rien n'indique qu'il s'agit de la circonférence du cercle dont nous avons entré le diamètre. Là encore, ce n'est pas très grave, puisque nous savons ce dont il s'agit ; mais si notre programme imprimait de nombreux résultats, il serait absolument indispensable de les identifier.

Comment, donc, faire imprimer un message indiquant la nature du nombre affiché ? Cela se fait très simplement grâce à l'instruction PRINT. Nous savions déjà faire afficher un texte, en l'entourant par des guillemets ; en faisant suivre le texte entre guillemets par une virgule,

elle-même suivie de la variable, nous obtiendrons le résultat voulu.

Dans notre cas, on pourra taper par exemple :

```
30 PRINT "LA CIRCONFERENCE DU CERCLE EST ", C
```

Dès lors notre programme est beaucoup plus aisé à utiliser : tapons RUN.

Il apparaît :

```
QUEL EST LE DIAMETRE DU CERCLE ?
```

Tapons un diamètre de 10 ;

Appuyons sur [RETURN] ; on obtient :

```
LA CIRCONFERENCE DU CERCLE EST 31.4159265

Ready
```

Si nous voulons déterminer la circonférence d'un autre cercle, il suffit de retaper RUN et le programme se réexécutera.

2.3.2 Instruction GOTO

Néanmoins, si nous avions beaucoup de circonférences à calculer, ce serait plus agréable de ne pas avoir à taper RUN à chaque fois. Une nouvelle instruction BASIC va nous offrir cette possibilité : l'instruction GOTO.

Tapons :

```
40 GOTO 10
```

L'effet de cette ligne, ajoutée en fin de programme, est de dire à Oric : "va à la ligne numéro 10" (en anglais "go to" signifie "allez à").

Que va-t-il alors se passer ? Après avoir calculé la circonférence et avoir imprimé le résultat, Oric arrive à cette ligne 40, qui lui dit de retourner à la ligne 10, ce qu'il fait. Il se retrouve donc en début de programme et recommence à l'exécuter, c'est-à-dire qu'il affiche à nouveau :

QUEL EST LE DIAMETRE DU CERCLE ?

C'est exactement ce que nous voulions obtenir !

Une telle structure, où un groupe d'instructions est répété plusieurs fois, s'appelle une boucle. C'est une notion très importante, dont nous verrons d'autres variantes, et qui permet à l'ordinateur d'effectuer des calculs itératifs.

2.3.3 Commande LIST

Notre programme ne comporte que 4 lignes, et il est donc aisé de s'en souvenir. Mais pour des programmes plus longs il sera indispensable de pouvoir faire imprimer les instructions telles qu'elles sont stockées en mémoire. C'est ce que permet la commande LIST, qui fait "lister" le programme en mémoire. Essayons et tapons :

```
LIST
```

Au stade où nous sommes nous obtenons :

```
10 INPUT "QUEL EST LE DIAMETRE DU CERCLE" ; D
20 C = PI * D
30 PRINT "LA CIRCONFERENCE DU CERCLE EST ", C
40 GOTO 10

Ready
```

Cette commande LIST sera revue ultérieurement et complétée. Pour le moment, revenons à notre programme. Il a encore un très gros défaut, il ne s'arrête jamais ! Indéfiniment il affichera :

QUEL EST LE DIAMETRE DU CERCLE ?

En fait , notre programme est un "mauvais programme",

car il comporte une boucle indéfinie, ce qui ne devrait jamais se produire. Un bon programme doit être prévu pour se terminer de lui-même au bout d'un temps fini.

Nous verrons par la suite comment mettre en place des boucles dont on soit sûr qu'elles se terminent.

2.3.4 CTRL C - CONT

En attendant, il nous faut savoir comment arrêter notre programme. Supposons que l'exécution soit en cours et qu'Oric demande :

QUEL EST LE DIAMETRE DU CERCLE ?

Si à ce moment là, nous avons terminé avec le calcul des circonférences, il suffira, pour stopper l'exécution, d'appuyer simultanément sur la touche [CTRL] et la touche [C].

Oric affiche alors :

Ready

Nous avons réussi à stopper l'exécution !

[CTRL] [C] est utile dans tous les cas où un programme ne s'arrête pas de lui-même ; mais son effet est légèrement différent lorsqu'on l'emploie alors que le programme n'est pas en attente sur INPUT (par exemple si le programme est en train d'effectuer de longs calculs). On obtient :

BREAK IN numéro de ligne

Ready

Le numéro de ligne qui apparaît après BREAK IN est celui de la ligne où en était le programme lorsqu'on a tapé [CTRL] [C].

Pour relancer l'exécution, on peut bien sûr taper RUN, mais aussi :

CONT (abréviation de CONTINUER)

Le programme redémarre alors à partir de la ligne où a eu lieu le BREAK et non pas, comme avec RUN, depuis le début.

Une autre différence avec RUN est que les valeurs des variables sont conservées. On peut donc, par exemple, après un BREAK, faire afficher les variables en mode direct pour connaître l'état du programme, puis reprendre l'exécution où elle en était par CONT.

ATTENTION : Si l'on apporte une modification au programme, la reprise de l'exécution devra impérativement se faire par RUN (et on recommencera donc depuis le début). Si on tape CONT, on aura le message d'erreur :

? CAN'T CONTINUE ERROR

2.3.5 RESET

Dans certains cas, pourtant, [CTRL] [C] se révèle inopérant ; on n'arrive plus à reprendre la main.

On peut alors bien sûr débrancher Oric, puis le rebrancher, mais tout ce qui était en mémoire est perdu, et en particulier le programme, qu'il faut alors retaper.

Une solution moins brutale consiste à appuyer sur la touche RESET. Vous ne la trouverez pas sur le clavier : c'est un petit bouton carré, noir, situé sur l'autre face d'Oric. Pour y appuyer, il faut un crayon ou tout autre objet effilé, car ce bouton est en renfoncement.

Son effet est le suivant : toute exécution en cours est arrêtée, toutes les variables sont remises à zéro ; l'appareil est ramené à son état initial, l'état où il se trouve lorsqu'on vient de le brancher, avec la différence, néanmoins, que le programme est toujours en mémoire.

Avant de poursuivre et découvrir de nouvelles instructions et commandes, nous allons récapituler ce que nous avons appris sur les instructions INPUT et PRINT, en ajoutant quelques compléments, car ces deux instructions sont fondamentales. Ce sont les commandes d'entrées-sorties par excellence, celles qui permettent le mieux de communiquer avec l'ordinateur, et à l'ordinateur de nous répondre.

2.3.6 Instruction INPUT

. Une ligne d'instruction de la forme :

numéro de ligne INPUT variable

provoque l'apparition d'un point d'interrogation, avec mise en attente du programme jusqu'à ce qu'on tape un nombre. La valeur tapée sera alors affectée à la variable.

. Pour préciser la nature du nombre à entrer, on peut ajouter un texte :

numéro de ligne INPUT "texte" ; variable

Le texte s'imprime alors, suivi d'un point d'interrogation.

. Il est possible d'entrer plusieurs variables dans un même INPUT, avec la syntaxe suivante :

numéro de ligne INPUT "texte" ; variable 1, variable 2

Exemple : Imaginons un programme qui nécessiterait l'entrée d'une date (par exemple pour calculer le nombre de jours entre deux dates). Il pourrait comporter l'instruction suivante :

numéro de ligne INPUT "ANNEE, MOIS, JOUR" ;
A, M, J

Notons les virgules entre les variables, et le point-virgule qui sépare le texte des variables.

A l'exécution, on obtiendra :

ANNEE, MOIS, JOUR ?

Il faudra alors taper 3 nombres, séparés par une virgule :

ANNEE, MOIS, JOUR ? 1789, 7, 14

et ne pas oublier d'appuyer sur [RETURN], comme d'habitude.

. Que se passe-t-il si au lieu de taper trois nombres, nous n'en tapons que deux et appuyons sur [RETURN] en oubliant le troisième ? Eh bien, Oric va prendre en compte les deux valeurs, s'apercevoir qu'il lui en faut encore une, et afficher un double point d'interrogation pour signaler qu'il manque des données.

```
ANNEE, MOIS, JOUR ? 1789, 14 [RETURN]
?? 7
```

On peut donc rentrer les données groupées comme l'on veut car, tant qu'Oric n'aura pas le nombre de données nécessaire, il affichera ??.

. Si au contraire nous tapons plus de données que nécessaire, Oric ne prendra en compte que les n premières (s'il attend n données) et ignorera les suivantes (superflues) en signalant le fait par le message EXTRA IGNORED.

```
ANNEE, MOIS, JOUR ? 1789, 14, 7, 18, 2
EXTRA IGNORED
```

Si nous ne tapons aucune donnée et faisons directement [RETURN], Oric, s'apercevant qu'aucune donnée n'a été tapée, nous demandera d'entrer une valeur, en affichant un point d'interrogation.

```
ANNEE, MOIS, JOUR ? [RETURN]
?
```

Si nous refaisons [RETURN], nous aurons un nouveau ? jusqu'à ce que nous tapions un nombre.

ATTENTION : Si nous faisons [RETURN] après un ??, Oric considérera que nous avons tapé 0.

2.3.7 Instruction PRINT

Pour imprimer une variable :

```
PRINT B
```

Pour imprimer un texte :

```
PRINT "texte"
```

Pour imprimer une variable en précisant sa nature :

```
PRINT "texte", B
```

On peut faire imprimer plusieurs variables par un même ordre PRINT, mais on a deux possibilités : on peut séparer les variables soit par des virgules, soit par des points-virgules. Quelle est la différence ?

Si l'on sépare deux variables par une virgule, la deuxième sera imprimée séparée de quatre blancs de la première variable. Si on utilise un point-virgule les deux variables seront imprimées côte à côte, séparées par un seul blanc.

Essayons et tapons :

```
B = 7
C = 13
PRINT B, C
7    13
PRINT B; C
7 13
```

Le principe est le même avec des chaînes de caractères ; la seule différence est que deux chaînes de caractères séparées par un point-virgule seront imprimées jointivement.

Par exemple :

```
PRINT "BON" ; "JOUR"
```

donne

```
BONJOUR
```

mais

```
PRINT "BON", "JOUR"
```

donne

```
BON   JOUR
```

De plus, il est possible de préciser le nombre de blancs entre deux chaînes de caractères.

Si l'on veut par exemple que BON et JOUR soient séparés par douze blancs, on écrira :

```
PRINT "BON" SPC(12) "JOUR"
```

Il suffit donc d'intercaler SPC(N) entre deux chaînes de caractères pour qu'elles soient séparées par N blancs. N doit être un nombre compris entre 0 et 255.

Pour en finir avec l'ordre PRINT, rappelons que l'on peut utiliser ? comme abréviation et signalons enfin que l'on peut terminer un ordre PRINT par un point-virgule, alors qu'il n'y a rien à séparer. Le prochain PRINT écrira alors sur la même ligne, comme le montre l'exemple suivant :

```
10 PRINT "BON";
20 PRINT "JOUR"
RUN
```

donne

```
BONJOUR

Ready
```

2.4 COMMANDES FONDAMENTALES

Avant de découvrir d'autres programmes, de tailles plus importantes, il est indispensable d'aborder les différentes commandes qui permettent la manipulation de ces programmes.

Nous avons déjà vu la commande LIST, qui permet de "lister" un programme, et sur laquelle nous allons revenir. Mais un programme doit aussi pouvoir être corrigé, modifié ; on doit pouvoir en chercher les erreurs... etc. Nous verrons les commandes qui permettent tout cela.

2.4.1 Connaître le contenu d'un programme : la commande LIST

Cette commande, que nous avons déjà rencontrée, fournit sur l'écran la copie du programme en mémoire, qu'on appelle la liste, ou en franglais le **listing**.

Si le programme est très long et ne tient pas tout entier sur l'écran, le "listing" va défiler, une ligne disparaissant en haut de l'écran, une nouvelle apparaissant en bas. La vitesse de défilement étant assez rapide, la lecture est assez difficile.

Aussi peut-on stopper le défilement en appuyant sur la barre d'espacement du clavier. Ensuite en appuyant sur n'importe quelle touche, le défilement reprend. Vous pouvez essayer, mais les programmes que nous avons écrits jusqu'à maintenant sont un peu courts pour que cela se voie bien.

En "listant" un programme, on constate que les instructions apparaissent dans l'ordre des numéros croissants, c'est-à-dire dans le même ordre que pour l'exécution, quel que soit l'ordre dans lequel elles ont été tapées. De plus, si en tapant nos instructions nous avons utilisé ? à la place de PRINT, sur le listing c'est le mot PRINT qui apparaît.

Au lieu de lister un programme d'un seul coup, il est possible de n'en visualiser que certaines parties :

Ainsi, pour lister une ligne x donnée, il suffit de préciser ce numéro après LIST. Exemple (nous supposons notre programme de calcul de circonférence toujours en mémoire) :

```
LIST 20
20 C = PI * D
```

Pour lister toutes les instructions comprises entre deux numéros x et y, il suffit de taper LIST x-y :

```
LIST 20-30
20 C = PI * D
30 PRINT "LA CIRCONFERENCE DU CERCLE EST ", C
```

Si l'on avait tapé LIST 25-38 on aurait obtenu la même chose, car, dans notre exemple, il n'y a aucune ligne entre 25 et 20, ni 30 et 38.

Pour lister toutes les lignes du début jusqu'à la ligne x il faut taper LIST -x :

```
LIST -20
10 INPUT "QUEL EST LE  DIAMETRE DU CERCLE"; D
20 C = PI * D
```

De la même façon LIST x- liste toutes les lignes de la ligne x jusqu'à la fin :

```
LIST 30-
30 PRINT "LA CIRCONFERENCE DU CERCLE EST ", C
40 GOTO 10
```

Si l'on veut arrêter définitivement de lister un programme en train de défiler sur l'écran, il faut appuyer simultanément sur [CTRL] et [C], comme pour arrêter l'exécution d'un programme. Il apparaît alors :

```
BREAK

Ready
```

2.4.2 Pour effacer un programme : commande NEW

Supposons qu'un programme soit en mémoire. Si nous tapons NEW en mode direct, tout le contenu de la mémoire sera effacé. Nous pouvons nous en assurer en utilisant la commande LIST : si après NEW nous tapons LIST, nous obtenons simplement :

```
Ready
```

preuve qu'aucun programme ne se trouve plus en mémoire.

Il est conseillé d'utiliser la commande NEW avant de taper un nouveau programme, pour effacer l'ancien.

En effet, imaginons que nous commencions à taper un nouveau programme sans avoir effacé l'ancien ; deux choses peuvent alors se produire : soit le numéro de la ligne que nous tapons existait déjà dans l'ancien programme, auquel cas la ligne que nous tapons ira remplacer l'ancienne, qui sera par là même effacée, soit ce numéro n'était pas dans l'ancien programme. Dans ce cas la ligne ira s'intercaler à la place impliquée par son numéro.

Nous comprenons donc que si les numéros des lignes que nous tapons ne correspondent pas un à un à ceux de l'ancien programme, il restera des instructions anciennes

non effacées, qui perturberont l'exécution du programme. D'où l'intérêt de la commande NEW, qui supprime cet inconvénient.

2.4.3 Pour lancer l'exécution d'un programme : commande RUN

Nous avons déjà rencontré la commande RUN qui permet de lancer l'exécution d'un programme. Il est possible d'utiliser cette commande sous la forme RUN x où x est un numéro de ligne, le programme s'exécutant à partir de cette ligne numéro x.

Il existe une autre façon d'obtenir le même résultat: on peut taper, en mode direct :

```
GOTO x
```

et là aussi le programme commencera à partir de la ligne x.

Il y a pourtant une différence : avec RUN, toutes les variables sont automatiquement mises à zéro, ce qui n'est pas le cas avec GOTO.

Exemples : tapons

```
10 B = 45
20 PRINT B
```

Alors	RUN		RUN 20	(B)		B = 7	(B)
				(est)		GOTO 20	(reste)
				(mis à)			(à)
				(0)			(7)
donne	45	donne	0		donne	7	

Il est possible de remettre les variables à zéro en tapant CLEAR.

Dans l'exemple précédent,

```
B = 7
CLEAR   (B est mis à 0)
GOTO 20
```

donne 0

2.4.4 Pour terminer un programme : instructions STOP et END

Jusqu'à maintenant nos programmes (du moins ceux qui n'avaient pas de boucles indéfinies) se terminaient en arrivant à leur dernière ligne.

Mais supposons que notre programme comporte plusieurs parties et que nous voulions lancer tantôt l'une tantôt l'autre, grâce par exemple à RUN x ; si nous lançons la première partie, rien ne signale à ORIC la fin de cette partie. Il va donc continuer en exécutant la deuxième partie.

Pour éviter cela, il faut terminer chaque partie par STOP ou END.

Ces deux instructions ont un effet sensiblement identique : en arrivant sur END ou STOP, le programme se termine et revient en mode direct. La seule différence est qu'avec END il s'affiche simplement

Ready

alors qu'avec STOP il s'affiche

BREAK IN numéro de ligne

Ready

Le numéro est celui de la ligne où se trouve le STOP. S'il y a plusieurs STOP dans le programme, on peut ainsi savoir où il s'est arrêté.

Traditionnellement, l'instruction END est plutôt placée à la toute dernière ligne de programme. Le END n'a pas plus de raison d'être à la fin qu'au début d'un programme. Cela dépend uniquement de la structure organique du programme.

2.4.5 Corrections et modifications de programmes

- INSERTION D'UNE LIGNE

Si nous voulons insérer une instruction entre les lignes 50 et 60 par exemple, il suffit de taper :

55 instruction

L'instruction viendra s'insérer à la ligne 55, entre les lignes 50 et 60. Mais nous voyons que cela n'est possible que parce que les numéros des lignes ne sont pas consécutifs. C'est pour cela qu'on a l'habitude (et c'est une bonne habitude à prendre !) de taper un programme en numérotant les lignes de 10 en 10 ; cela permet les insertions ultérieures. Naturellement, on pourrait aussi numéroter de 20 en 20 ou de 100 en 100, mais l'expérience montre que la numérotation de 10 en 10 est la plus pratique.

- SUPPRESSION D'UNE LIGNE

Comme nous l'avons déjà vu, il suffit de taper son numéro, suivi de [RETURN].

- MODIFICATION COMPLETE D'UNE LIGNE

Nous savons aussi que pour modifier complètement une ligne, il suffit de la retaper, précédée de son numéro. L'ancienne version est effacée et remplacée par la nouvelle.

- MODIFICATION DE QUELQUES CARACTERES SUR UNE LIGNE

Supposons qu'en faisant lister notre programme nous nous apercevions que nous avons fait une erreur dans une ligne, par exemple :

```
30 PRINT "LA CIRCONGERENCE DU CERCLE EST ", C
```

Une première solution consiste à retaper entièrement la ligne, mais c'est fastidieux si la ligne est un peu longue.

La bonne solution est la suivante : il faut d'abord faire lister la ligne défectueuse sur l'écran, par exemple ici par :

```
LIST 30
■ 30 PRINT "LA CIRCONGERENCE DU CERCLE EST ", C
```

Il faut ensuite amener le curseur au début de la ligne comme indiqué ci-dessus, grâce aux touches de mouvement du curseur :

[←] [↓] [↑] [→]

On appuie simultanément sur [CTRL] [A]. Le curseur avance alors vers la droite ; on l'amène ainsi jusqu'au caractère erroné, ici le G :

```
30 PRINT "LA CIRCON■ERENCE DU CERCLE EST ", C
```

(Pour cela on maintient la touche [CTRL] enfoncée et l'on appuie sur [A] autant de fois que nécessaire pour placer le curseur).

On tape alors un F, puis on réappuie sur [CTRL] [A] pour faire parcourir au curseur le reste de la ligne :

```
30 PRINT "LA CIRCONFERENCE DU CERCLE EST ", C■
```

On appuie ensuite sur [RETURN] : la modification est alors prise en compte.

ATTENTION : L'effet de la touche [CTRL] [A] est de placer dans une petite mémoire intermédiaire les caractères rencontrés sur l'écran par le curseur. Si nous avions fait [RETURN] après avoir tapé le F, les caractères suivants n'auraient pas été pris en compte. En faisant

```
LIST 30
```

nous aurions obtenu

```
30 PRINT "LA CIRCONF
```

S'il y a plusieurs caractères erronés sur la même ligne, il suffit de les corriger de la même façon, au fur et à mesure que le curseur se positionne sur ces caractères.

2.5 CREEZ VOTRE JEU EN BASIC

2.5.1 Présentation du jeu

Jusqu'à maintenant nous avons vu quelques instructions fondamentales du BASIC, et les commandes indispensables à l'élaboration des programmes. Néanmoins, nous ne sommes pas capables d'écrire des programmes très élaborés, car notre vocabulaire BASIC est encore assez limité.

Dans cette partie nous allons découvrir les instructions BASIC supplémentaires qui, avec celles que nous connaissons déjà, forment véritablement les bases de la programmation. Pour cela, nous allons programmer un jeu qui nous permettra d'introduire de manière plus attrayante de nouvelles instructions BASIC.

Au départ, notre jeu sera bien sûr un peu simple, mais nous l'améliorerons progressivement. A quoi allons-nous donc jouer ? Eh bien, nous allons essayer de programmer le jeu de la "roulette russe"... ou tout au moins sa simulation.

Rappelons le principe de ce jeu (si dangereux dans la réalité) : un revolver à six coups contient une seule cartouche ; on fait tourner le barillet au hasard une fois, puis chacun des joueurs à tour de rôle braque l'arme contre sa tempe et appuie sur la détente, jusqu'à ce que l'un d'eux renonce, auquel cas il a perdu, ou jusqu'à ce que le coup parte...

Pour simuler le coup de feu, nous pourrons par exemple faire écrire BANG. Pour repérer la balle dans le barillet, nous pourrons utiliser une variable qu'on appellera par exemple BALLE, et qui vaudra un nombre de 1 à 6. On affichera alors BANG au premier coup si BALLE vaut 1 (c'est-à-dire si la balle est dans le premier logement du barillet), au deuxième coup si BALLE = 2, au troisième si BALLE = 3, etc... Il nous faudra donc compter les coups ; pour cela nous introduirons une variable COUP qui vaudra 1 au premier coup et que nous augmenterons de 1 à chaque coup.

Mais il faudra aussi que nous puissions tester si BALLE = COUP ; il existe pour cela une instruction BASIC, l'instruction IF, que nous allons détailler :

2.5.2 Instruction IF

Sa forme générale est la suivante :

IF condition THEN instruction 1 ELSE instruction 2

Si la condition est vraie (**IF** signifie SI), on exécute l'instruction 1 qui suit le THEN (**THEN** signifie ALORS). Si la condition est fausse on effectuera l'instruction 2 qui suit ELSE (ELSE veut dire SINON).

Puis on passe à la ligne suivante du programme.

Précisons que l'instruction ELSE n'est pas obligatoire ; on peut écrire simplement :

```
IF condition THEN instruction 1
```

Si la condition est fausse, on passe à la ligne suivante, sans effectuer l'instruction 1.

Exemple

```
50 IF A = B THEN PRINT "A EGALE B" ELSE PRINT
   "A EST DIFFERENT DE B"
```

La condition qui est testée ici est A = B. La machine compare les valeurs de A et B ; si ces deux valeurs sont égales, l'instruction qui suit THEN est effectuée : Oric écrit A EGALE B. Sinon, il effectue l'instruction 2, c'est-à-dire qu'il écrit A EST DIFFERENT DE B.

Lorsque l'instruction qui suit THEN est GOTO, la syntaxe est susceptible de se simplifier : on peut toujours écrire :

```
70 IF A > B THEN GOTO 110
```

mais aussi simplifier par

```
70 IF A > B THEN 110
```

ou encore

```
70 IF A > B GOTO 110
```

On comprend aisément la signification de cet exemple. Si la valeur de A est supérieure à la valeur de B, on va à la ligne 110, sinon on passe à la ligne 80, suivante.

L'instruction IF est peut-être la plus importante des

instructions BASIC ; aussi, avant de poursuivre, examinons en détail les types de condition qui peuvent suivre le IF.

- Un premier type de condition est de la forme :

Expression arithmétique	Opérateur de comparaison	Expression arithmétique

Exemples :

A	=	B
A	>	B
3 * A - 2	>	4 * B ↑ 2
---	---	---
Expression arithmétique	Opérateur de comparaison	Expression arithmétique

Les opérateurs de comparaison sont les suivants :

= égal
> supérieur
< inférieur
< > différent
> = supérieur ou égal
< = inférieur ou égal

- Un deuxième type de condition combine des conditions du premier type à l'aide des trois opérateurs logiques suivants :

AND : et
OR : ou
NOT : non

La syntaxe est la suivante (condition A et condition B sont des conditions du premier type) :

condition A AND condition B

est vrai si, et seulement si, condition A et condition B sont vraies toutes les deux.

condition A OR condition B

est vraie si l'une au moins des deux conditions A ou B est vraie (elles peuvent aussi, naturellement, être vraies toutes les deux).

NOT condition A

est vraie si condition A est fausse et fausse si condition A est vraie.

Exemple

```
70 IF 2 * A > 4+3↑C OR B< 8 THEN 30
```

Cette instruction enverra à la ligne 30 si 2 * A est strictement supérieur à 4+3↑C ou si B est strictement plus petit que 8. Si aucune de ces deux conditions n'est vérifiée, on passera à la ligne suivante.

Nous avons dit que IF est une des plus importantes instruction BASIC. En effet, c'est elle qui permet à Oric de prendre des décisions, c'est-à-dire d'exécuter une instruction plutôt qu'une autre, en fonction des différentes situations résultant des données ou des calculs.

Nous sommes donc, maintenant, en mesure de taper une première version de notre jeu.

2.5.3 Première version du jeu

```
10 CLS
20 BALLE = 4
30 COUP = 1
40 PRINT "CLIC..."
50 IF COUP = BALLE THEN PRINT "BANG" ELSE
   PRINT "SAUVE !"
60 INPUT "TAPEZ 1 POUR TIRER 0 POUR
   ABANDONNER"; REP
70 IF REP = 0 THEN END
80 COUP = COUP + 1
90 GOTO 40
```

Nous découvrons à la ligne 10 une nouvelle instruction, l'instruction CLS. Elle a simplement pour but de nettoyer l'écran de tout le texte qui peut s'y trouver et

de ramener le curseur en haut à gauche de l'écran. On peut aussi la taper en mode direct, mais il est alors plus simple de taper [CTRL] [L] qui a le même effet.

La suite du programme se comprend aisément ; au départ COUP = 1 : c'est le premier coup ! On affiche CLIC... pour suggérer que l'on commence à appuyer sur la détente. Puis, grâce à l'instruction IF, on teste si la balle est dans le logement du barillet face au canon, c'est-à-dire pour nous si BALLE = COUP. Si c'est le cas on imprime BANG : le coup est parti ; sinon on imprime SAUVE !

On demande ensuite si l'on veut tirer à nouveau. Si l'on tape 0 le jeu s'arrête, si l'on tape 1 (en fait tout nombre autre que 0), le jeu continue : on va tirer une nouvelle fois et donc passer au coup suivant. Pour cela on augmente COUP de 1, à la ligne 80. Cette instruction ne doit pas surprendre. En mathématique, bien sûr, elle serait fausse, mais ici elle signale simplement que l'on ajoute 1 à la variable COUP, puis que cette nouvelle valeur est affectée à la même variable COUP.

Ensuite, on repart à la ligne 40 où l'on recommence à "tirer".

Une première objection vient tout de suite à l'esprit. A la ligne 20 il y a BALLE = 4. Nous savons donc où se trouve la balle : dans le quatrième logement du barillet. Le jeu serait satisfaisant pour quelqu'un qui n'aurait pas écrit le programme et qui ignorerait donc combien vaut BALLE. Mais il est toujours possible de le découvrir en listant le programme.

Il existe un moyen d'éviter cet inconvénient : faire tirer un nombre au hasard par l'ordinateur lui-même.

2.5.4 Fonction RND

Cela est possible grâce à la fonction RND (abréviation du mot anglais **RANDOM**, qui signifie hasard).

Si nous tapons :

```
PRINT RND(1)
```

nous obtenons un nombre aléatoire compris entre 0 et 1.

Cela peut surprendre : comment un ordinateur dont le comportement est parfaitement déterminé, peut-il donner des nombres au hasard ?

En fait ces nombres ne sont pas vraiment tirés au hasard. Ils sont calculés par des algorithmes précis, mais qui, paradoxalement, donnent des suites de nombres, dits pseudo-aléatoires, et qui sont tels que, statistiquement, tout se passe comme s'ils avaient été tirés au hasard.

Un appel à RND est de la forme RND (X) avec X > 0 : on obtient alors un nombre entre 0 et 1. Des appels successifs avec la même valeur de X donnent les éléments successifs d'une même suite pseudo-aléatoire (le nombre obtenu changeant à chaque fois). En changeant la valeur de X on change de suite, mais généralement on utilise RND (1).

2.5.5 Fonction INT

Néanmoins notre problème n'est pas encore résolu, car nous voulons un nombre entier entre 1 et 6. Pour cela, nous allons utiliser la fonction INT qui donne la valeur entière immédiatement inférieure.

Ainsi :

```
PRINT INT (3.1416)
```

donne 3

```
PRINT INT (127.4387)
```

donne 127

Attention : pour les nombres négatifs, on obtient

```
INT (-3.1416) = -4
INT (-127.4387) = -128
```

Comme nous voulons un nombre entier de 1 à 6, nous taperons :

```
20 BALLE = INT (6 * RND(1)) + 1
```

RND(1) est compris entre 0 et 1, 0 et 1 exclus, donc 6 * RND(1) est compris entre 0 et 6, 0 et 6 exclus, d'où INT (6 * RND(1)) est compris entre 0 et 5, 0 et 5 inclus. En ajoutant 1, on a bien un nombre entier de 1 à 6.

Notre programme n'est pas encore parfaitement satisfaisant, car une fois que le coup est parti, c'est-à-dire qu'on a imprimé BANG, il demande encore si l'on veut tirer, alors que, normalement, le jeu est terminé.

Pour éviter cet inconvénient nous pourrions, bien sûr, mettre à la place de PRINT "BANG" un GOTO vers une instruction faisant imprimer BANG, suivie de l'instruction END. Mais il existe une meilleure façon de faire : l'utilisation de l'instruction REPEAT ... UNTIL.

2.5.6 REPEAT ... UNTIL

Cette instruction s'emploie de la façon suivante :

```
REPEAT
  :
  :
instructions
  :
  :
UNTIL condition
```

REPEAT signifie "répéter", **UNTIL** signifie "jusqu'à ce que". Toutes les lignes d'instructions comprises entre REPEAT et UNTIL sont répétées, c'est-à-dire réexécutées jusqu'à ce que la condition suivant UNTIL soit vraie. Cette condition est du même type que celles qui peuvent suivre un IF que nous avons déjà vu.

Cette structure pourrait être remplacée par IF NOT condition GOTO x à la place de UNTIL, x étant le numéro de la première ligne des instructions à répéter, mais le programme serait beaucoup moins clair à lire et à comprendre. Aussi l'instruction REPEAT ... UNTIL est-elle à utiliser partout où c'est possible.

2.5.7 Seconde version du jeu

Nous allons donc pouvoir modifier notre programme de la façon suivante :

```
10  CLS
20  BALLE = INT (6 * RND(1)) + 1
30  REPEAT
40  WAIT 100
50  INPUT "TAPEZ 1 POUR TIRER 0 POUR
    ABANDONNER"; REP
60  IF REP = 0 THEN END
70  PRINT "CLIC..."
80  WAIT 200
85  COUP = COUP + 1
90  IF BALLE < > COUP THEN PRINT "SAUVE !"
100 UNTIL BALLE = COUP
110 PRINT "BANG"
120 EXPLODE
130 END
```

Nous avons aussi ajouté deux instructions nouvelles, l'instruction WAIT et l'instruction EXPLODE.

2.5.8 L'instruction WAIT

En anglais, **WAIT** signifie ATTENDRE ; WAIT 100 aura pour effet d'arrêter l'exécution pendant 100 fois 10 millisecondes, soit 1 seconde, WAIT 200 pendant 200 fois 10 millisecondes, soit 2 secondes... ; de manière générale, une ligne d'instruction WAIT N provoquera un arrêt de l'exécution de N fois 10 millisecondes.

Nous avons ajouté cette instruction pour laisser un intervalle de temps entre 2 impressions sur l'écran, et aussi pour ajouter au suspense, par l'attente du BANG ou du SAUVE ! après le CLIC... !

2.5.9 L'instruction EXPLODE

Cette instruction provoque un bruit d'explosion, qui donne beaucoup plus d'effet au jeu. Vous pouvez essayer de la taper en mode direct.

Il existe d'autres instructions provoquant des bruits

divers, tels SHOOT, ZAP, PING ; ces instructions sont vues plus en détail dans le chapitre réservé aux possibilités sonores d'Oric.

Avant de conclure, nous allons présenter une instruction assez semblable à REPEAT ... UNTIL, et elle aussi très fondamentale : la boucle FOR ... NEXT.

2.5.10 Boucle FOR ... NEXT

REPEAT ... UNTIL nous permet de répéter une action jusqu'à ce qu'une condition soit vérifiée, mais, a priori, nous ne savons pas combien de fois les instructions seront répétées.

Aussi, pour pouvoir répéter une suite d'instruction N fois, il existe l'instruction FOR ... NEXT, qui s'emploie de la façon suivante :

```
FOR I = 1 TO N
  :
  :
instructions
  :
  :
NEXT I
```

Son fonctionnement est le suivant : au départ I = 1, et les instructions sont exécutées. On arrive à l'instruction NEXT I (qui signifie I suivant), où l'on doit tester si I est égal ou non à N. Si I est plus petit que N, I est augmenté de 1 et l'on réexécute les instructions, puis l'on compare à nouveau I et N.

Les instructions sont ainsi répétées jusqu'à ce que I = N, auquel cas on passe à la ligne suivant le NEXT.

Cette instruction FOR ... NEXT sera revue et complétée dans le chapitre suivant, aussi contentons-nous de donner ici un exemple simple : faire imprimer les nombres de 1 à 1000.

```
10 FOR I = 1 TO 1000
20 PRINT I
30 NEXT I
```

REMARQUE : Le nom de la variable, qui ici est I, peut bien sûr être différent.

2.6 CONCLUSION

Nous avons pu, dans ce chapitre, découvrir un premier jeu sur Oric, jeu assez simple, mais qui nous laisse entrevoir que les possibilités n'ont de limites que celles de notre imagination.

Parvenus à la fin de ce livre, vous serez capables d'écrire vous-même vos programmes de jeux. Dans cette perspective, le chapitre suivant va nous permettre d'approfondir encore notre connaissance du BASIC, en récapitulant les instructions que nous avons déjà vues et en nous en présentant de nouvelles.

CHAPITRE **3**

Le langage Basic de l'oric

3.1 GENERALITES

L'objet de ce chapitre est de développer et de préciser les notions dont vous avez eu un aperçu au chapitre précédent.

Nous allons examiner successivement les instructions constituant le langage évolué, compréhensible par votre microordinateur, qui vous permettront de réaliser tous vos programmes.

- LE SYSTEME DE CARACTERES EMPLOYE :

Le but d'un programme est, en général, de calculer des "valeurs". En fait, un ordinateur ne "connaît" ces valeurs que sous forme de représentations. Celles-ci sont, comme nous l'avons vu au chapitre précédent, des configurations physiques constituées de bits, d'octets, de "paquets" d'octets.

Pour représenter les caractères qu'il comprend, votre Oric utilise le code de représentation ASCII (initiales d'**American Standard Code of Information Interchange**).

Vous trouverez la liste des caractères et leur code ASCII correspondant dans l'annexe F.

Vous pouvez noter que votre ordinateur utilise 1 octet (8 bits) pour représenter chaque caractère. Le mode de représentation est globalement le suivant :

les valeurs

0 à 31	caractérisent les caractères de contrôle (CTRL T ... CTRL N etc...)
32 à 127	caractérisent les caractères alphanumériques et les caractères spéciaux du code ASCII.
128 à 255	caractérisent les mots constituant le vocabulaire BASIC compréhensible par votre Oric.

- LES CARACTERES ALPHANUMERIQUES SONT :

. caractères alphabétiques :
A, B, ..., Z (alphabet majuscule)

. caractères numériques :
0, 1, ..., 9

- LES CARACTERES SPECIAUX SONT :

+ - * / ( ) . , ; " = ' blanc $ % # ! a, b, c, ..., z (alphabet minuscule) etc...

3.2 CONSTANTES - VARIABLES - INDICES

3.2.1 Les constantes

Votre microordinateur distingue deux grands types de constantes : les constantes numériques et les constantes alphanumériques (ou chaînes de caractères).

- LES CONSTANTES NUMERIQUES

Celles-ci sont des nombres décimaux, de signe quelconque, qui peuvent s'écrire de trois façons distinctes :

. une série de chiffres sans point décimal (nombre entier).

exemple : PRINT 7653

. une série de chiffres avec point décimal (nombre réel dit en virgule fixe).

exemple : PRINT 76.535

. un entier ou un réel en virgule fixe suivi de la lettre E, d'un signe - éventuellement - et d'un entier (nombre réel dit en virgule flottante).

exemple : PRINT 76.535 E+23

Le symbole E s'énonce "dix puissance" et signifie ci-dessus que 76.535 est multiplié par 10^{23} soit
$\underbrace{10.....0}_{23 \text{ zéros}}$

Toutes ces constantes sont considérées par votre Oric comme des nombres réels. La représentation de ces réels s'effectue dans la machine à l'aide de 5 octets.

1 octet est utilisé par l'exposant
(dans l'exemple précédent +24)

4 octets sont utilisés pour la mantisse
(dans l'exemple précédent 7.653500)

Vous trouverez des renseignements complémentaires sur la représentation interne des réels en annexe E.

Ce type de représentation vous imposera de limiter vos réels entre $-1.70141\ 10^{38}$ et $1.70141\ 10^{38}$; les réels compris entre $-2.93874\ 10^{-39}$ et $+2.93274\ 10^{-39}$ sont assimilés à 0.

Il est à noter qu'une constante supérieure à $1.70141\ 10^{38}$ ou inférieure à $-1.70141\ 10^{38}$ provoque l'apparition du message d'erreur ? OVERFLOW ERROR (dépassement de capacité de la machine). De même l'écriture d'une constante dont la mantisse possède plus de 38 chiffres entraîne l'affichage du même message d'erreur.

Considérons maintenant :

```
PRINT 11.123456789012345 [RETURN]
11.123456789
```

Observez qu'Oric n'édite que 9 chiffres après le point décimal, ceci s'appelle une troncature.

En outre, compte tenu de la représentation interne des réels, la machine a une précision de l'ordre de 9 chiffres sur la mantisse.

Pour nous convaincre, réalisons le programme suivant.

```
 5 INPUT"A";A
10 INPUT"B";B
15 C=(A=B)
20 PRINT"A=";A,"B=";B
25 IF   C=0   THEN   PRINT   "ORIC   CONSIDERE
   QUE";A;"EST DIFFERENT DE";B
30 IF   C=-1   THEN   PRINT   "ORIC   CONSIDERE
   QUE";A;"EST EGAL A";B
35 END
```

Vous pouvez prendre, par exemple, dans un premier temps :

```
          A=1.123456789  et B=1.123456790
puis      A=1.1234567890 et B=1.1234567891
```

Il est important que vous gardiez en mémoire que la précision de calcul de votre Oric est de 9 chiffres après la virgule sur la mantisse.

La conclusion que l'on peut tirer de cet exemple est que si la machine accepte des mantisses de 38 chiffres, seuls les dix premiers sont significatifs. Donc, lors de l'entrée des données, il est inutile de taper des mantisses de plus de 9 chiffres.

Oric admet aussi une écriture des constantes entières en numération en base 16 (numération hexadécimale).

exemple : #10, # -E100

C'est le symbole ♯ qui indique à la machine que les caractères qui suivent figurent un nombre hexadécimal. (Ce nombre doit évidemment être écrit à l'aide de chiffres hexadécimaux).

Ainsi : 5 A =♯ G
10 PRINT A
RUN
provoque l'affichage du message d'erreur ? SYNTAX ERROR IN 5 (G n'est pas un chiffre hexadécimal).

Vous trouverez des renseignements complémentaires sur la numération hexadécimale en annexe E.

Si vous tapez par exemple, PRINT ♯ 1EFA [RETURN], vous noterez qu'Oric écrira directement ce nombre en base décimale. La conversion inverse, c'est-à-dire l'expression en base hexadécimale d'un nombre décimal, s'effectue à l'aide de la fonction HEX$.

Exemple : PRINT HEX$(16)

Notez que la conversion de la base 16 à la base 10 est limitée à des entiers variant entre - ♯FFFF et + ♯FFFF (-65535 et +65535).

Ainsi PRINT ♯ 10000 provoque l'affichage du message OVERFLOW ERROR.

- LES CONSTANTES ALPHANUMERIQUES

Celles-ci sont constituées d'une suite de n caractères du code ASCII, situés entre cotes "doubles".

Exemple : PRINT "je m'appelle Oric"

Notez que la double cote " servant à délimiter une chaîne de caractères ne peut pas être un caractère appartenant à cette chaîne.

Si la chaîne de caractères est la dernière donnée écrite sur une ligne, en mode direct ou différé, la double cote " finale peut être oubliée sans inconvénient, puisque l'action sur la touche [RETURN] s'y substitue. Toutefois l'emploi de cette solution pour écrire une chaîne de caractères n'est pas à conseiller.

La longueur minimale d'une constante alphanumérique est zéro (chaîne vide) ; sa longueur maximale est de 255 caractères.

La représentation interne d'une chaîne s'effectue par le codage de chaque caractère, un à un, par le code ASCII donc elle nécessite un octet par caractère de chaîne.

3.2.2 Les variables simples

Une variable est une entité mathématique à laquelle on peut affecter diverses valeurs "variables". Pour définir une variable, il suffit de lui donner un nom.

Ce nom doit entrer dans une syntaxe bien précise à laquelle vous devez vous conformer strictement.

. Le nom de la variable doit commencer par un caractère alphabétique (A, B, ..., Z) et peut être suivi par des caractères alphanumériques (A, ..., Z, 0, ..., 9). Les caractères $ et % peuvent aussi être utilisés, mais ils ont un rôle précis : ils caractérisent le type de la variable (variable alphanumérique ou entière). Ces deux caractères se placent toujours à la fin du nom de la variable.

. Le nom d'une variable doit comprendre au maximum 238 caractères. En fait, l'interpréteur basic de votre Oric représente chaque variable par les deux premiers caractères de son nom, et par son type. Ainsi, les deux variables POIRE et POIDS sont considérées comme identiques pour la machine.

Toutefois, ce n'est pas une raison pour que vous limitiez, dans vos programmes, les noms de vos variables à 2 caractères. En effet, il est intéressant, pour améliorer la lisibilité de vos programmes, de prendre comme nom de variable le nom du résultat du calcul qu'on lui affecte, ou bien le nom de ce que la variable représente.

Expliquons ceci plus clairement sur un exemple simple.

Construisons un programme calculant l'aire d'un cercle connaissant son rayon.

```
 5 REM CALCUL DE L'AIRE D"UN CERCLE
   CONNAISSANT SON RAYON
10 INPUT "QUEL EST LE RAYON DU CERCLE";RAYO
15 AIRE = PI * RAYO * RAYO
20 PRINT "L'AIRE DU CERCLE DE RAYON " ; RAYO
   ; "VAUT" ; AIRE
25 END
```

Ainsi la variable qui représente le rayon du cercle s'appelle RAYO, et celle à laquelle on affecte le calcul de la surface du cercle s'appelle AIRE.

Notez que PI est une constante stockée par Oric ; cette constante vaut 3.14159265.

. En dernier lieu le nom d'une variable ne doit, en aucun cas, contenir de mot appartenant au vocabulaire basic de la machine (la liste de ces mots "réservés" figure dans l'annexe A).

Ainsi, des noms de variable aussi anodins que PRUNE, VALEUR, FORCE, LETTRE, NEUTRON, etc ne doivent pas être employés.

L'emploi d'un mot réservé dans un nom de variable provoque généralement l'apparition du message d'erreur SYNTAX ERROR. Sinon l'interpréteur exécutera sans réaction immédiate le mot réservé, ce qui aura des conséquences aussi imprévisibles que variées.

Vous pouvez utiliser trois types de variables dans vos programmes : les variables alphanumériques, entières, ou réelles.

- LES VARIABLES ALPHANUMERIQUES OU CHAINES DE CARACTERES

Une variable est alphanumérique si le dernier caractère de son nom est le symbole $. Le "contenu" d'une telle variable doit obligatoirement être une constante alphanumérique.

Exemple : AA$= "A=1 + BCDXY 56"

Si vous essayez AA$=1234567890, l'interpréteur basic vous signalera l'erreur par le message TYPE MISMATCH ERROR (les cotes sont fondamentales puisque ce sont elles

qui indiquent que la constante est une chaîne de caractères).

Il est à noter que la représentation interne de cette variable AA$ s'effectue de la façon suivante : Oric stocke dans cet ordre :

. le nom de la variable (AA$), ce qui occupe 2 octets (les deux premiers caractères du nom sont seuls concernés, le symbole $ est pris en compte dans la manière de stocker la variable) ;

. la longueur de la chaîne de caractères, ce qui occupe 1 octet (ceci vous explique pourquoi une chaîne comprend au plus 255 caratères $255 = 2^8-1$) ;

. l'adresse de la zone où est stockée la chaîne de caractères, ce qui occupe 2 octets ;

. deux octets "séparateurs".

Donc finalement, une variable alphanumérique occupe 7 octets + le nombre d'octets correspondant au nombre de caractères composant la chaîne. En conclusion, vous devez retenir que l'emploi de variables alphanumériques est coûteux en place mémoire.

- LES VARIABLES ENTIERES

Une variable est entière si le dernier caractère du nom de celle-ci est le symbole %. Le "contenu" d'une telle variable doit obligatoirement être une constante non alphanumérique.

Exemple :
```
 5 CA%=123
10 PRINT CA%
RUN
```

Remarquez que CA% est un nombre entier (absence de point décimal).

A présent, modifiez la ligne 5 :

```
5 CA%=1535.428
RUN
```

Avec cet exemple vous pouvez voir que lorsque l'on affecte à CA% une valeur réelle, la machine stocke en fait le plus grand nombre entier inférieur à la valeur réelle que vous avez affectée à CA%. (D'un point de vue mathématique, Oric prend la partie entière de ce nombre réel).

Afin d'améliorer la lisibilité de vos programmes futurs, il est souhaitable que, dès à présent, vous preniez l'habitude d'affecter à une variable entière une constante mathématiquement de même type.

Ainsi, plutôt que de taper : CA%=13.22,
il est préférable d'utiliser : CA%=INT(13.22)
(la fonction INT est la fonction partie entière, ces deux lignes sont donc équivalentes).

REMARQUE : CA%=#FAB est bien entendu licite puisque #FAB est un entier en numération hexadécimale. Notez que PRINT CA% écrit CA% en numération décimale (en base 10).
Si vous essayez CA%="123", l'interpréteur basic vous signalera l'erreur par le message TYPE MISMATCH ERROR, puisque la présence de " " signifie que vous affectez à CA% une constante alphanumérique.

La représentation interne de la variable CA% s'effectue de la façon suivante.

Oric stocke dans cet ordre :

. le nom de la variable (CA%) ce qui occupe 2 octets (les 2 premiers caractères du nom sont seuls conservés, le symbole % est pris en compte dans la façon de stocker la variable) ;
. la valeur affectée à la variable, ce qui occupe 2 octets.

Donc, le stockage d'une variable entière n'occupe que 4 octets. Etant donné que la valeur de la variable est stockée sur 2 octets seulement, vous ne pourrez utiliser que les entiers compris entre -32768 et +32767.

Vous trouverez des renseignements complémentaires sur la représentation des entiers en annexe.

- LES VARIABLES REELLES

C'est l'option par défaut que prend la machine lorsque le nom de la variable ne possède comme dernier caractère ni $, ni %. Le contenu de cette variable doit être une constante non alphanumérique.

Exemple : DA=1.23 E23
DA=3452.34.

La représentation interne de la variable réelle DA s'effectue de la façon suivante :

Oric stocke dans cette ordre :

. le nom de la variable, ce qui occupe 2 octets (les 2 premiers caractères du nom sont seuls conservés) ;

. la valeur affectée à la variable, ce qui, vous l'avez vu au paragraphe précédent, nécessite 5 octets.

Le stockage d'une variable réelle n'occupe donc que 7 octets.

Pour mémoire, les réels que vous devez utiliser sont compris entre $-1.70141\ 10^{38}$ et $1.70141\ 10^{38}$. Bien entendu, l'écriture d'une ligne du type CA="123" entraîne l'affichage du message suivant : TYPE MISMATCH ERROR.

En conclusion, vous pouvez remarquer que suivant le type de variable que vous considérez, telle ou telle représentation physique sera adoptée par la machine. C'est pourquoi les variables A, A$ et A%, de noms pourtant peu différents sont, en fait, fondamentalement distinctes.

Il est donc capital de préciser à la machine, au moment de la programmation, le type de chaque variable que vous voudrez utiliser. Cette gymnastique se révèlera profitable à la longue, car elle vous conduira à définir tous les "objets" que vous voudrez employer et, donc, à mieux les contrôler.

A présent, étudiez le petit exercice suivant :

Une personne de vos amis peut effectuer ses achats dans 5 magasins différents. Mais elle veut connaître celui qui est, en moyenne, le meilleur marché. Pour ce faire, elle se constitue un panier de référence de 26 articles tests tous disponibles dans chaque magasin et regarde le prix de chaque "panier de la ménagère".

En outre, inquiète de l'évolution des prix, elle veut faire des essais tous les mois. Elle sait que vous possédez un ordinateur et elle fait appel à vous pour élaborer un programme permettant d'effectuer chaque mois les calculs nécessaires à la réalisation de son enquête.

Soucieux d'utiliser les conseils prodigués précédemment, vous définissez logiquement les variables qui vont contenir les prix de chaque objet de chaque magasin. Vous appelez les objets A, B, ..., Z (26 objets) et les magasins 1, ..., 5 (5 magasins), les noms des variables prix seront constitués du nom de l'objet auquel elles se rapportent et du nom du magasin où a été acheté l'objet, elles auront donc pour nom A1, B1, ..., Z1, A2..., Z2, ..., Z5.

Les prix de chaque panier seront les variables AA, BB, ..., EE (il faut se garder de donner le même nom à deux variables différentes) ; par exemple le prix du panier 3 sera CC = A3 + B3 + ... + Z3.

Mais avant de faire ce calcul, il faut affecter à chaque objet son prix dans chaque magasin (cette opération porte le nom générique de "saisie de données").

Votre programme débute par :

```
 5 REM CALCUL DU MAGASIN LE MEILLEUR MARCHE
10 INPUT "PRIX DE L'ARTICLE 1 DU MAGASIN
   1";A1
15 INPUT "PRIX DE L'ARTICLE 2 DU MAGASIN
   1";B1
```

Vous vous rendez compte à ce moment que vous allez taper cette ligne pour chaque article de chaque magasin, soit 130 fois, ce qui est extrêmement fastidieux. Or chaque variable A1, B1, ... Z5 a les mêmes propriétés.

En fait, votre Oric permet de résoudre et d'écrire beaucoup plus élégamment ce genre de problèmes constitués

de travaux, comme ici, très répétitifs, c'est-à-dire la construction de variables de mêmes caractéristiques et leur emploi dans les calculs.

Dans le paragraphe suivant, nous allons justement voir comment construire simplement un groupe de variables de mêmes propriétés.

3.2.3 Les variables indicées ou tableaux

Votre ami étant une personne organisée, vous a en fait remis le document suivant pour vous faciliter la saisie des données.

Tableau des prix des articles dans chaque magasin.

(Tableau I)

article / magasin	A	B	J	Z
1	A1	B1	J1	Z1
2	A2	B2	J2	Z2
3	A3	B3	J3	Z3
4	A4	B4	J4	Z4
5	A5	B5	J5	Z5

Votre Oric est pratiquemment capable de construire et d'utiliser un tel document. D'un point de vue informatique ce document s'appelle un tableau, et les éléments de ce tableau s'appellent des variables indicées.

Voyons comment construire ce tableau.

Pour que Oric puisse connaître le tableau précédent, il suffit de lui donner un nom et de définir la structure et la taille de ce tableau. Ceci peut se réaliser par le mot DIM de la façon suivante :

```
5 DIM PRIX(4,25)
```

Explication : PRIX est le nom que nous avons décidé de donner au tableau contenant le prix de chaque article de chaque magasin.

Les règles de formation des noms de tableaux sont strictement les mêmes que celles concernant la formation des noms de variables "simples" vues au paragraphe précédent et que nous pouvons rappeler brièvement :

- le premier caractère du nom du tableau doit être exclusivement alphabétique (A, ..., Z majuscule).

- les autres caractères du nom sont alphanumériques : (A, ..., Z, 0, ..., 9).

- les noms de tableaux doivent comporter au plus 238 caractères, mais en fait Oric n'utilise que les deux premiers pour caractériser le nom.

- le nom du tableau ne doit pas contenir de mots "réservés" du vocabulaire basic de la machine.

Voyons à présent pourquoi le nom PRIX est suivi par deux nombres entre parenthèses. Le document remis par votre ami est un tableau qui comporte des lignes et des colonnes, c'est ce que l'on appelle un tableau à deux dimensions :

. une dimension "horizontale" puisqu'il y a des lignes.
. une dimension "verticale" puisqu'il y a des colonnes.

Physiquement cette notion de dimension se traduit par le nombre minimal d'informations qu'il faut fournir pour retrouver un élément du tableau. Par exemble, l'élément J3 du tableau est à l'intersection de la troisième ligne et de la dixième colonne, donc 2 renseignements.

Le nombre de dimensions d'un tableau est ce que nous avons appelé structure du tableau.

Les parenthèses qui suivent le mot PRIX sont en fait les deux caractères qui indiquent à votre Oric qu'il est en présence d'un élément de tableau. Et il y a autant de nombres dans les parenthèses qu'il y a de dimensions dans le tableau.

Votre Oric vous permet de construire des tableaux possédant entre une seule et 88 dimensions. La seule limitation est en fait due à la taille de la mémoire de votre ordinateur.

Ceci nous amène tout naturellement à parler de la taille des tableaux et de la façon de repérer un élément dans son tableau.

Lorsque nous avons écrit DIM PRIX(4,25), Oric a réservé la place nécessaire pour stocker le tableau PRIX, et pour lui le tableau a la structure suivante :

article / magasin	0	1		9		25
0	A1	B1				
1	A2	B2				
2	A3	B7		J3		
3	A4	B4				
4	A5	B5				

Les éléments de la ligne "article" qui se substituent aux lettres A, ..., Z du tableau I sont appelés indices de colonne.

Les éléments de la colonne "magasin" qui se substituent aux numéros des magasins du tableau sont appelés indices de ligne.

Chaque indice de colonne repère une colonne différente et chaque indice de ligne une ligne différente.

Pour repérer la "case" dans ce tableau PRIX où est rangé J3, il suffit de donner le nom du tableau, puis le numéro de la ligne où se trouve J3 , c'est-à-dire l'indice de ligne, ici 2,et le numéro de la ligne où se trouve J3 , c'est-à-dire l'indice de colonne, ici 9.

Donc la case du tableau PRIX contenant J3 est la variable indicée PRIX(2,9).

Notez, et ceci est fondamental, que votre Oric fait commencer les indices à 0.

En fait, lors de l'écriture DIM PRIX(4,25), on donne à Oric la variable indicée, c'est-à-dire l'élément du tableau PRIX, qui possède les numéros d'indices les plus grands.

Ceci nous amène à dire que votre microordinateur ne connaît finalement le tableau PRIX que par l'intermédiaire de ses éléments, les variables indicées.

A la limite, une variable indicée peut être interprêtée comme une variable "simple" ou "élémentaire", et de la même façon qu'il admet des variables entières, alphanumériques ou réelles, Oric admet des tableaux de variables indicées entières, alphanumériques, ou réelles.

- TABLEAUX DE VARIABLES ENTIERES

Oric reconnaît que toutes les variables indicées d'un même tableau sont entières lorsque le dernier caractère du nom du tableau est le symbole %.

Exemple : 5 DIM A%(50)

Toutes les variables indicées de A%(0) à A%(50) doivent être entières et comprises entre -32768 et +32767.

Si vous écrivez :

```
10 A%(0)=1.255
15 PRINT A%(0)
```

vous constaterez qu'Oric écrira 1, c'est-à-dire qu'il donne le plus grand entier inférieur à 1.255 (comme pour les variables entières, il prend la partie entière).

Bien que la machine accepte l'écriture précédente, il est préférable de toujours affecter à une variable indicée entière, un entier ; ceci peut être réalisé à l'aide de la fonction INT (partie entière) éventuellement.

Pour stocker dans ses mémoires un tableau d'entiers, Oric utilise :

. deux octets contenant les 2 premiers caractères du nom du tableau ;

. deux octets contenant l'adresse du tableau suivant ;

. un octet contenant le nombre de dimensions du tableau ;

. deux octets par dimension contenant la valeur du plus grand indice dans la dimension ;

. deux octets par variables indicées entières.

Voyons ceci sur un exemple.

Le tableau B%(10,10,10,10) nécessite :

pour stocker ses caractéristiques :

$2 + 2 + 1 + 2 \times \underbrace{4}_{(4 \text{ dimensions})} = 13$ octets

pour stocker la valeur de chaque élément.

$\underbrace{(11 \times 11 \times 11 \times 11)}_{\text{nombre d'éléments dans le tableau}} \times 2 = 29282$ octets

Notez en comparaison que votre Oric dispose au maximum de 49.152 octets de RAM en code machine, et de 39.421 pour l'usage de la RAM en BASIC.

- TABLEAUX DE CHAINES DE CARACTERES

Oric reconnaît que toutes les variables indicées d'un même tableau sont alphanumériques lorsque le dernier caractère du nom du tableau est le symbole $.

Exemple : 5 DIM B1$(50,50)

Toutes les variables indicées de B1$(0,0) à B1$(50,50) doivent contenir des chaînes de caractères.

Si vous tapez 10 B1$(25,10)=23.4 l'interpréteur basic vous signalera l'erreur par le message TYPE MISMATCH ERROR.

Pour stocker dans ses mémoires un tableau de chaînes de caractères, Oric utilisera :

. deux octets contenant les 2 premiers caractères du nom du tableau ;

. deux octets contenant l'adresse du tableau suivant ;

. un octet contenant le nombre de dimensions du tableau ;

. deux octets par dimension contenant la valeur du plus grand indice dans la dimension ;

. trois octets contenant les paramètres d'identification de chaque variable indicée (adresse en zone mémoire et longueur de la chaîne de caractères affectée à la variable).

Si chaque élément du tableau B1$ contient une chaîne de 10 caractères, votre microordinateur a besoin de 2+2+1+2x2+3x(51x51)=7812 octets pour stocker les caractéristiques du tableau B1$.

Il ne faut pas oublier que pour chaque chaîne de caractères, Oric utilise, pour la stocker, un nombre d'octets égal au nombre de caractères de cette chaîne, donc il a besoin de 51x51x10=26010 octets pour stocker chaque chaîne de caractères affectée au tableau B1$.

Donc Oric utilise 33822 octets pour stocker B1$.

Vous devez retenir de cet exemple que l'emploi de tableaux de variables alphanumériques peut nécessiter une place mémoire très importante même si ceux-ci ne comportent qu'un petit nombre de variables.

- TABLEAUX DE VARIABLES REELLES

C'est l'option par défaut que la machine prend lorsque le nom du tableau ne possède pas comme dernier caractère $ ou %.

Exemple : 5 DIM BA(100,50)

Toutes les variables indicées de BA(0,0) à BA(100,50) devront être réelles et comprises entre $-1.70141\ 10^{38}$ et $1.70141\ 10^{38}$.

Pour stocker dans ses mémoires un tableau de réels, Oric utilise :

. deux octets contenant les 2 premiers caractères du nom du tableau ;

. deux octets contenant l'adresse du tableau suivant ;

. un octet contenant le nombre de dimensions du tableau ;

. deux octets par dimension contenant la valeur du plus grand indice dans la dimension ;

. cinq octets par variable indicée réelle.

Donc le tableau BA(100,50) nécessite :

- pour stocker ses caractéristique 2+2+1+2x2=9 octets
- pour stocker la valeur de chaque variable réelle 101x51x5=25755 octets.

Soit en tout 25764 octets.

A noter en comparaison que le tableau BA%(100,50) de mêmes dimensions utilise 10311 octets. Ceci devrait achever de vous convaincre de la nécessité qu'il y a de préciser exactement à la machine le type des variables et des tableaux de variables que vous voulez utiliser.

Avant de passer au paragraphe suivant qui va vous rappeler l'utilisation des opérations arithmétiques vues au chapitre précédent, il reste un dernier point à préciser à propos des variables indicées.

Les indices doivent être entiers et strictement positifs ou nuls.

En utilisant les notions vues au chapitre précédent et les notions sur les variables indicées, vous êtes en mesure de réaliser le programme "calcul du magasin le meilleur marché".

Essayez !

3.3 LES OPERATEURS

Un opérateur est un outil qui permet d'éxécuter des opérations mathématiques ou logiques. Oric distingue trois types d'opérateurs, ainsi que vous l'avez vu, les opérateurs arithmétiques, relationnels et logiques (ou booléens).

3.3.1 Les opérateurs arithmétiques

. l'opérateur d'addition	noté +
. l'opérateur de multiplication	noté *
. l'opérateur de soustraction	noté -
. l'opérateur de division	noté /
. l'opérateur d'élévation à la puissance	noté ↑

Nous vous rappelons que "A élevé à la puissance B", qui s'écrit A↑B signifie que l'on multiplie A par lui-même B fois.

Ainsi écrivons :

```
? 5↑3 [RETURN]
125
```

B n'est pas nécessairement entier mais A est nécessairement un nombre réel strictement positif.

Si vous tapez :

```
 5 A=-2
10 PRINT A↑1.2
RUN
```

Le basic vous signalera l'erreur par le message ILLEGAL QUANTITY ERROR IN 10.

Bien entendu, les opérateurs arithmétiques n'agissent que sur des nombres entiers ou réels, le résultat étant un nombre qu'Oric considère comme réel.

3.3.2 Les opérateurs relationnels

- l'opérateur de stricte supériorité noté >
  (A > B A strictement supérieur à B)

- l'opérateur de stricte infériorité noté <
  (A < B A strictement inférieur à B)

- l'opérateur d'égalité noté =
  (A = B A est égal à B)

- l'opérateur de supériorité noté > =
  (A ≥ B A supérieur ou égal à B) ou = >

- l'opérateur d'infériorité noté < =
  (A ≤ B A inférieur ou égal à B) ou = <

- l'opérateur de non égalité noté > <
  (A ≠ B A différent de B) ou < >

Vous pouvez remarquer à cet égard la différence fondamentale de nature entre le signe = opérateur d'égalité et le signe = qui signifie que l'on affecte à une variable le résultat d'une opération arithmétique. Le basic de votre Oric possède une instruction qui permet de "marquer" cette différence : c'est l'ordre LET.

Exemple : LET A=1.5+5*4.23

L'ordre LET assigne à A le résultat de l'opération 1.5+5*4.23. Il est facultatif, aussi LET A=5 et A=5 signifient la même chose. Cette signification du signe = vous permet de comprendre pourquoi l'expression

```
5 A=A+1
```

est licite, et ce qu'elle signifie ; à la variable A on affecte le résultat du calcul de l'addition A+1.

Rappelons que l'instruction

```
10 C=A=B
```

signifie qu'à la variable C on affecte le résultat du "test" A=B.

Dans un test réalisé avec n'importe lequel des opérateurs relationnels,

si la réponse à ce test est positive C vaut -1,
si la réponse à ce test est négative à C est affectée la valeur 0.

Donc le résultat d'un test (c'est ainsi que nous appellerons désormais une expression relationnelle) est un entier qui vaut 0 ou -1.

REMARQUE : Pour améliorer la lisibilité de la ligne 10 précédente, c'est-à-dire au niveau de l'écriture, bien différencier les deux rôles du signe =, vous pouvez adjoindre des parenthèses.

```
10 C=(A=B)
```

Le rôle des parenthèses utilisées dans des expressions arithmétiques ou logiques sera reprécisé plus loin.

3.3.3 Les opérateurs logiques

. l'opérateur AND qui signifie ET

. l'opérateur OR qui signifie OU

. l'opérateur NOT qui prend la négation.

Les opérateurs logiques, aussi appelés booléens, agissent sur des tests (ou expressions logiques ou relationnelles).

- ACTION DE AND

AND s'utilise ainsi : A AND B, avec A et B expressions relationnelles ; l'expression globale A AND B constitue elle aussi un test. Le résultat du test A AND B est positif si la réponse à chacun des deux tests A et B est positive.

- ACTION DE OR

OR s'utilise ainsi : A OR B, avec A et B tests ; l'expression globale A OR B constitue elle aussi un test.

Le résultat du test A OR B est positif si l'une au moins des réponses, à l'un des deux tests A ou B, est positive.

- ACTION DE NOT

NOT s'utilise ainsi : NOT A, avec A test. L'expression globale NOT A constitue elle aussi une expression logique. Le résultat du test NOT A est l'opposé du résultat du test A.

Ainsi, si la réponse au test A est "vrai", alors la réponse au test NOT A est "faux" et réciproquement.

3.3.4 Règles d'évaluation des expressions arithmétiques et logiques

L'expression 2+3*4 peut s'interpréter de deux façons, soit (2+3)x4=20, soit 2+(3x4)=14.

Pour lever cette ambiguïté Oric utilise des règles d'évaluation, c'est-à-dire que certaines opérations seront exécutées avant d'autres : cela signifie qu'il y a une hiérarchie naturelle des opérateurs.

L'ordre de priorité décroissant des opérateurs est le suivant :

↑, NOT, * /, + -, tous les opérateurs relationnels, AND, OR.

Exemple : PRINT 2↑3+2*3 [RETURN]
14

Oric a effectué cette opération de la façon suivante: Calcul de 2↑3(=8) puis de 2*3 (=6) puis finalement de (2↑3)+(2*3)=14.

Mais si vous, vous vouliez qu'Oric calcule : (2↑(3+2))*3, il suffit d'afficher cette opération sur l'écran de télévision comme elle est écrite sur cette feuille. Ce sont les parenthèses qui permettent de modifier à volonté l'ordre des opérateurs.

Lorsque vous utiliserez des parenthèses dans les opérations arithmétiques ou logiques, les opérations situées à l'intérieur des parenthèses seront effectuées avant celles qui sont à l'extérieur des parenthèses.

Si vous avez plusieurs niveaux de parenthèse (parenthèses imbriquées) ce sont les opérations situées dans les parenthèses les plus "intérieures" qui seront effectuées les premières, et ainsi de suite. Vérifiez toujours que vous avez autant de parenthèses ouvrantes "(" que de parenthèses fermantes ")", sinon votre Oric vous signalera l'erreur par le message SYNTAX ERROR.

Sur l'exemple précédent (2↑(3+2))*3, il y a deux niveaux de parenthèses :

$$\underbrace{(\ 2\ \uparrow\ \underbrace{(\ 3 + 2\ )}_{\text{2ème niveau}}\ )\ *\ 3}_{\text{1er niveau}}$$

Maintenant que nous avons défini les "objets" qui seront manipulés dans vos programmes, il ne nous reste plus qu'à voir les ordres basic qui vont constituer la charpente de ceux-ci.

3.4 LES INSTRUCTIONS BASIC FACULTATIVES

Nous avons choisi d'appeler ainsi certaines instructions basic qui, si elles sont employées, ne sont pas prises en compte, ou bien si elles sont absentes, amènent la machine à prendre une option par défaut. Leur emploi est tout de même fondamental puisqu'elles améliorent de façon considérable la lisibilité du programme.

3.4.1 CLEAR

Cet ordre permet de remettre toutes les variables numériques à zéro et les variables alphanumériques à la chaîne "vide". Cette instruction n'a pas d'utilité en début de programme, puisque l'ordre RUN que vous utilisez poour lancer le programme initialise automatiquement toutes les variables à zéro ou au vide.

Avant d'utiliser cette instruction vérifiez que toutes les valeurs calculées dans les instructions précédant CLEAR ne vous sont plus d'aucune utilité après cet ordre. (Ne pas mettre CLEAR dans les boucles FOR ... NEXT et REPEAT ... UNTIL par exemple).

3.4.2 DIM

Cette instruction que nous avons déjà rencontrée permet de dimensionner un tableau. Elle est facultative. En effet si vous écrivez comme première instruction :

```
5  BEC(1,1,1) = 7
```

Oric considère que BEC est un tableau de réels et que la variable indicée de BEC qui possède les indices les plus grands est BEC(10,10,10).

BEC possède donc 11x11x11=1331 éléments.

Donc, si vous n'utilisez pas DIM, à la première variable indicée qu'il trouvera, Oric allouera un tableau dont la variable de plus haut niveau d'indice sera 10 dans chaque dimension.

Lorsqu'un tableau est dimensionné, il ne peut être redimensionné dans la suite du programme.

Ainsi :

```
 5 DIM A(30)
10 DIM A(35)
RUN
```

provoque l'affichage du message REDIM'D ARRAY ERROR. De même :

```
 5 A(1)=0
10 DIM A(35)
```

provoque l'affichage du même message.

Ceci s'explique parce qu'Oric, rencontrant la variable indicée A(1) à la première ligne du programme, considère par défaut que A est un tableau de 11 éléments. La ligne suivante est alors considérée comme une tentative de redimensionnement du tableau A.

L'ordre DIM doit précéder toute instruction où un élément du tableau dimensionné par DIM est utilisé, pour être licite.

Vous devez vous souvenir que lorsque vous écrivez DIM A(35), cela signifie que A(35) est l'élément du plus haut indice du tableau A, et que le tableau A possède 36 éléments A(0), ..., A(35).

3.4.3 END

Cette instruction provoque l'arrêt du programme, mais contrairement à l'ordre STOP que nous verrons ultérieurement, n'indique pas le numéro de la ligne où s'est effectué l'arrêt du programme.

Cette instruction n'est pas obligatoire, car l'exécution s'arrête lorsque la dernière instruction physique du programme est rencontrée (sauf si cette instruction est un ordre de transfert, ordre que nous verrons au paragraphe suivant).

3.4.4 FRE

Cette instruction permet de connaître en mode direct ou différé, le nombre d'octets encore disponibles dans la mémoire de votre microordinateur.

Exemple : ? FRE (0)

En mode différé elle a un autre avantage qui peut être très intéressant.

Considérons :

```
 5 A$="ABCDE"
10 A$="123456"
15 PRINT A$
```

Pour exécuter ce programme, votre Oric stockera :

à la ligne 5 :	A, B, C, D, E, l'adresse du premier caractère A, la longueur de la chaîne A$.
à la ligne 10 :	A, B, C, D, E, 1, 2, 3, 4, 5, 6, l'adresse du premier caractère de la chaîne A$, c'est-à-dire 1 et la longueur de la chaîne A$.

Cet exemple vous montre que lors du maniement de chaînes de caractères Oric stocke toutes les chaînes intermédiaires de calculs, ce qui occupe beaucoup de place. Justement l'ordre FRE permet de remédier à ce problème, car il permet à votre Oric de libérer ses mémoires de ces bouts de chaînes inutiles et encombrants.

Il s'utilise sous la forme :

```
1030 A = FRE (" ")
```

Il permet, lors du maniement de chaînes de caractères, de "purger" le contenu des mémoires.

REMARQUE : Cet ordre est évidemment indispensable lors du maniement de grandes chaînes de caractères ou lorsqu'il y a beaucoup d'opérations effectuées sur ces chaînes. Par contre cet ordre demande un certain temps d'exécution.

3.4.5 LET

Cette instruction qui est facultative assigne une valeur à une variable.

Exemple : 5 LET A=3

Bien entendu, la variable et la constante doivent être de même type. (Si la variable est entière, la constante étant réelle, la valeur effective assignée à la variable est l'entier immédiatement inférieur.

3.4.6 STOP

Cette instruction provoque l'arrêt de l'exécution du programme.

Si vous tapez :

```
10 A=10
15 STOP
20 PRINT A
RUN
```

Oric répond :

```
BREAK IN 15
```

C'est-à-dire que, lors de l'utilisation du mot STOP, le programme s'arrête là où cet ordre est interprêté et il y a affichage du numéro de la ligne où se situe l'ordre qui a provoqué l'arrêt du programme.

3.4.7 REM

Cette instruction a pour seul but de vous permettre d'insérer des commentaires dans vos programmes.

Lorsque l'interpréteur basic rencontre cette instruction en début de ligne, celle-ci est considérée comme inexistante. Ainsi, si vous mettez plusieurs instructions basic sur la même ligne (ce qui est permis par le caractère :) où REM est le premier ordre, ces instructions ne seront pas prises en compte.

Exemple :
```
 5 REM PROGRAMME D'ESSAI : A=1 : PRINT A
10 STOP
RUN
```

Sur l'écran vous voyez s'afficher :

```
BREAK IN 10
```

Cette instruction est fondamentale car il ne faut pas oublier que la principale qualité d'un programme est, outre sa justesse bien entendu, sa lisibilité. Celle-ci permet aux utilisateurs et à l'auteur... de comprendre pas à pas ce qu'effectue le programme lors de sa lecture.

Toutes les instructions que nous venons de voir sont "facultatives". Elles ont un autre point commun dans la mesure où l'interpréteur basic les prend en compte suivant l'ordre croissant du numéro de la ligne dans laquelle elles se trouvent. Nous allons voir à présent des instructions qui permettent de modifier l'ordre de déroulement d'un programme, c'est ce que l'on appelle des ordres de transfert.

3.5 LES INSTRUCTIONS DE TRANSFERT

Le basic de votre microordinateur distingue deux types d'instruction de transfert.

3.5.1 Les ordres de transfert inconditionnels

- GOTO

Cet ordre indique, à l'interpréteur, d'aller exécuter

les instructions à partir d'une ligne donnée du programme.

Exemple :

```
 5 I=I+1
10 GOTO 5
15 END
RUN
```

Sur cet exemple vous pouvez voir que le nombre qui suit l'ordre GOTO (5) est entier, c'est le numéro de la ligne où l'on veut se brancher. Ce nombre est exprimé explicitement mais peut aussi être calculé comme dans l'exemple suivant :

```
 5 I = 20
10 GOTO I
15 STOP
20 PRINT I
25 I = I - 5
28 GOTO I
```

L'entier qui suit GOTO doit effectivement correspondre à un numéro de ligne existant dans le programme.

Eliminez les modifications précédentes et écrivez :

```
10 GOTO 30
RUN
```

Il n'y a pas de ligne 30 dans le programme, le message d'erreur UNDEF'D STATEMENT ERROR s'affiche sur l'écran.

Si vous revenez à l'exécution du programme initial, vous vous rendez compte qu'effectivement le programme se rebranche toujours à la ligne 5... et vous ne passez jamais par la ligne 15. Autrement dit ce programme ne peut pas s'arrêter de lui-même. Pour "reprendre la main" vous êtes obligé de taper [CTRL] [C] ou d'appuyer sur [RESET], touche située à l'opposé du clavier.

Cet exemple vous montre donc ce qu'est une boucle sans fin.

C'est en fait la chose dont il faut se méfier avec cette instruction GOTO. En général on remédie à ce problème en utilisant un ordre de transfert conditionnel (voir plus loin).

- ON ... GOTO

Cette instruction est, elle aussi, un ordre de transfert inconditionnel, mais elle s'emploie un peu différemment de GOTO.

Voyons ceci sur l'exemple suivant :

```
 5 REM EXEMPLE D'UTILISATION DE L'INSTRUCTION
   ON ... GOTO
10 INPUT "I=";I
15 ON I GOTO 25,30
20 STOP
25 PRINT "LORSQUE I=1 LE TRANSFERT S'EFFECTUE
   SUR"
26 PRINT "LA PREMIERE ETIQUETTE DU GOTO"
27 GOTO 10
30 PRINT "LORSQUE I=2 LE TRANSFERT S'EFFECTUE
   SUR"
31 PRINT "LA DEUXIEME ETIQUETTE DU GOTO"
35 GOTO 10
RUN
```

Pour I entrez successivement 1, 2, 3 et observez ce qui se produit, puis relancez le programme et entrez 0, 255, 256, -1.

Vous voyez donc que lorsque vous tapez ON I GOTO 25, 30, I se rapporte à la position du numéro de ligne sur laquelle on veut se brancher : I doit donc être un entier. Si vous entrez un réel, Oric effectuera l'opération qui vous est maintenant familière : il affecte à I le plus grand entier inférieur à ce réel.

Lorsque vous avez entré 256 ou -1, la machine répond ILLEGAL QUANTITY ERROR. Ceci vous prouve que I doit toujours être positif, d'une part, et d'autre part que vous n'avez droit qu'à 255 numéros de lignes derrière le GOTO.

Lorsque vous avez tapé 0 ou 3, vous vous êtes rendu compte que la machine s'arrêtait à la ligne 30, c'est-à-dire que si I est nul ou si I est plus grand que le nombre d'étiquettes derrière le GOTO ($I \leqslant 255$), l'instruction ON ... GOTO est ignorée et l'interpréteur passe à la ligne suivante. Bien entendu, les numéros de lignes situés derrière le GOTO sont entiers et explicités. Ils doivent correspondre à des numéros de lignes existants.

3.5.2 Les ordres de test ou de comparaison

- IF ... THEN ... ELSE

Cet ordre que vous avez vu au chapitre précédent appartient au deuxième type d'instruction de transfert : c'est un ordre de transfert conditionnel.

Rappelons, sur l'exemple vu pour l'instruction GOTO, le fonctionnement de IF ... THEN ... ELSE.

```
 5 I=I+1
10 PRINT I
15 IF (I > 1000) THEN STOP ELSE GOTO 5
RUN
```

Lors de l'exécution vous voyez défiler sur l'écran tous les nombres entiers de 1 à 1000 puis BREAK IN 15 ; que s'est-il passé ?

La ligne 15 se traduit littéralement par :

Si I > 1000 alors l'exécution s'arrête, sinon le programme se branche à la ligne 5 et c'est ce que permet de faire cette instruction.

Sa syntaxe est IF A THEN B ELSE C. A est un test (expression logique) B et C étant des instructions basic. Si la réponse au test A est "vrai", l'ordre B est exécuté sinon c'est l'ordre C qui est exécuté.

L'ordre ELSE est facultatif. S'il est absent et que le test A est faux le programme passe à la ligne suivante. Si le test A est vrai, l'ordre B et exécuté comme précédemment.

Si vous écrivez IF A THEN cette instruction de transfert n'a aucune action.

Si ELSE est utilisé, B ne devra pas comporter d'opérations arithmétiques, sinon il apparaîtra le message SYNTAX ERROR IN lorsque B sera exécuté.

Le programme suivant vous montre un exemple simple d'utilisation de l'instruction IF ... THEN ... ELSE.

```
10 REM SAISIE DE DONNEES
20 REM
30 INPUT "NOM DE L'ARTICLE";N$
40 INPUT "PRIX";PX
50 PRINT N$, PX
60 INPUT "VALIDEZ-VOUS CES DONNEES ";A$
70 IF ((A$<>"O") AND (A$<>"N")) THEN 60
   ELSE IF (A$="O") THEN PRINT "DONNEES
   ENTREES" ELSE 30
80 END
```

Faites exécuter ce programme plusieurs fois en tapant par exemple :

	DRAP
puis	70.
puis	N (non)
puis	O (oui)

à la question VALIDEZ-VOUS CES DONNEES ?

REMARQUE : Il ne peut y avoir dans une instruction basic plus de deux tests IF consécutifs. Par exemple, la ligne 70 du programme ci-dessus serait refusée, si vous remplaciez l'instruction PRINT "DONNEES ENREGISTREES" par l'instruction nouvelle IF (PX=0) THEN 40.

Nous avons donc vu les instructions basic qui permettent d'exécuter des instructions d'une ligne de programme, à partir d'une autre. Par ailleurs vous avez vu un exemple de "boucle" que l'on peut effectuer à l'aide de l'ordre IF ... THEN ... ELSE GOTO. En fait le basic de votre Oric a des commandes qui sont spécifiques à ce genre de cas ; ce sont ces instructions que nous allons examiner maintenant.

3.6 LES BOUCLES

Le basic de votre microordinateur possède deux instructions spécifiques et distinctes pour faire du calcul répétitif qu'on appelle boucles :

L'instruction FOR ... TO ... STEP ...
NEXT ...

et l'instruction REPEAT
UNTIL ...

Ces deux groupes d'ordres permettent de faire répéter autant de fois que l'on veut les instructions situées entre les deux lignes qui constituent la syntaxe des deux groupes d'instruction.

3.6.1 FOR ... TO ... STEP ... NEXT ...

Voyons l'utilisation de cette instruction sur un exemple :

```
 5 REM CALCUL DU MAGASIN LE MEILLEUR MARCHE
10 DIM PRIX(4,25) : DIM PANIER(4)
15 REM ENTREE DES DONNEES ET CALCUL DU PRIX
   DE CHAQUE PANIER
20 FOR I=0 TO 4 STEP 1
22 IP1=I+1
25:     FOR J=0 TO 25 STEP 1
27:     JP1=J+1
30:     PRINT "QUEL EST LE PRIX DU PRODUIT
        No";JP1;"DANS LE MAGASIN";IP1
33:     INPUT PRIX(I,J)
35:     PANIER(I)=PANIER(I)+PRIX(I,J)
40:     NEXT J
45:     NEXT I
50 REM RECHERCHE DU MAGASIN LE MEILLEUR
   MARCHE
55 K=PANIER(0) : L=0
60 FOR I=1 TO 4
65 IF (K> PANIER(I)) THEN L=I
70 IF (K> PANIER(I)) THEN K=PANIER(I)
72 NEXT I
73 LP1=L+1
75 PRINT "LE MAGASIN No";LP1;"EST LE MEILLEUR
   MARCHE CE MOIS-CI"
80 END
```

Examinons la syntaxe de cette instruction de boucle.

A la ligne 20, nous avons FOR I=0 TO 4 STEP1 et en 45 NEXT I. Ceci signifie que toutes les instructions situées entre les lignes 20 et 45 seront exécutées tant que le réel I (qui est appelé indice de la boucle) sera inférieur à 4. Ici 0 est la valeur initiale de I et 4 la borne supérieure de I. 1 est le nombre qui est ajouté à I chaque fois que l'instruction de la ligne 45 est exécutée (on dit que l'on incrémente I de 1 lorsque NEXT I est exécuté).

Le test de comparaison de I et de sa borne supérieure n'est effectué qu'à l'instruction NEXT I, ce qui signifie que, quelle que soit la valeur de I, les instructions qui sont dans la boucle seront effectuées au moins une fois.

Dans la ligne 60 vous voyez que STEP n'a pas été utilisé. En fait ce mot est facultatif ; lorsqu'il n'est pas employé la machine prend par défaut un pas égal à 1.

En résumé la syntaxe de l'instruction s'écrit :

```
FOR I=A TO B STEP C
  :
  :
NEXT I
```

A, B et C sont des nombres réels quelconques (C peut être négatif), ce sont les paramètres de la boucle. Lorsque I est supérieur ou égal à B (si C est positif) ou I inférieur ou égal à B (si C est négatif), la boucle est "désactivée"; c'est l'instruction qui est après NEXT I dans le programme qui est alors exécutée. Il est fortement déconseillé de modifier les paramètres de la boucle à l'intérieur de celle-ci.

Dans les lignes qui sont situées dans la boucle peut figurer n'importe quel ordre basic. Vous pouvez utiliser un ordre GOTO pour sortir d'une boucle. Mais ceci n'est pas recommandé car le basic, n'ayant jamais exécuté le test (I supérieur ou égal à B ou I inférieur ou égal à B) avec une réponse positive, considère que la boucle est toujours active bien que, physiquement, ce ne soit pas le cas.

Vous pouvez, bien entendu, mettre une instruction FOR ... TO ... STEP ... NEXT ... dans une boucle (voir ligne 25 à ligne 40 de l'exemple précédent) ; dans ces conditions on dit que l'on est en présence de boucles imbriquées. La syntaxe des boucles imbriquées est :

```
 10 FOR I=A1 TO B1 STEP C1
  :
100 FOR J=A2 TO B2 STEP C2
  :
200 NEXT J
  :
300 NEXT I
```

Les indices des deux boucles doivent avoir un nom différent, sinon la boucle extérieure est rendue inactive. Il n'y a aucun inconvénient à ce que deux boucles non imbriquées aient le même indice (cf exemple précédent). Il y a incrémentation sur l'indice extérieur à chaque fois que la boucle intérieure est "désactivée". On ne peut pas imbriquer plus de 10 boucles les unes dans les autres, sans obtenir le message NEXT WITHOUT FOR ERROR. Si vous laissez plus de dix boucles "actives" parce que vous "sortez" de celles-ci par l'ordre GOTO, le même message s'affichera.

La syntaxe

```
 10 FOR I=A1 TO B1 STEP C1
  :
100 FOR J=A2 TO B2 STEP C2
  :
200 NEXT I
  :
300 NEXT J
```

où, vous le remarquez, deux boucles se "croisent", est erronée. En présence de telles boucles, il y a affichage du message d'erreur NEXT WITHOUT FOR ERROR.

Considérons le problème suivant : on se propose de trouver des triplets de nombres entiers (A, B, C) A, B, C compris entre 1 et N tels que $A^2+B^2=C^2$ (côtés d'un triangle rectangle). Une solution classique de ce problème est le triplet (3, 4, 5). On peut écrire le programme suivant :

```
 5  REM RECHERCHE DES CARRES DE PYTHAGORE
    JUSQU'A UN CERTAIN RANG
 8  CLS
10  INPUT "VEUILLEZ ENTRER LA BORNE SUPERIEURE
    N";N
15  FOR A=1 TO N
17  A%=INT(A)
20: FOR B=1 TO N
23: B%=INT(B)
25:     FOR C=1 TO N
27:     C%=INT(C)
30:     IF ((A%*A%+B%*B%-C%*C%)=0) THEN
        PRINT A,B, C
35:     NEXT C
40: NEXT B
45  NEXT A
50  END
```

Vous remarquerez que l'exécution de ce programme est longue, surtout lorsque N est grand. La question que l'on peut se poser est : peut-on remédier à ce problème ?

Dans ce programme, on peut observer quelques points "litigieux".

Dans le test de la ligne 30 vous constatez que l'on calcule dans la boucle la plus intérieure les produits A%*A% et B%*B% à chaque incrémentation sur C. Or les deux variables A% et B% sont indépendantes de C. Par conséquent dans la boucle

```
25 FOR C=1 TO N
  :
35 NEXT C
```

A%*A% et B%*B% sont des constantes ; il est inutile de les recalculer dans cette boucle. Ces produits doivent donc être évalués à l'extérieur de la boucle.

Il faut donc retenir pour vos programmes futurs que lorsque vous utiliserez des boucles, il faut vérifier si à l'intérieur de celles-ci vous n'effectuez pas des opérations sur des variables indépendantes de l'indice de la boucle.

Pour ce programme spécifique, il est encore possible d'améliorer la vitesse d'exécution. En effet, il est inutile de faire jouer à A et B le même rôle puisque, par exemple, les triplets (3, 4, 5) et (4, 3, 5) sont en fait identiques. On décide que A sera inférieur à B, et $A^2+B^2=C^2$ montre que A et B sont strictement inférieurs à C (d'ailleurs A est alors strictement inférieur à B). Ceci permet de modifier les paramètres des diverses boucles, afin de diminuer le nombre d'itérations.

Ainsi : C variera de 1 à N
B variera de 1 à C-1
A variera de INT (SQR(2*C-1)) à B-1

Il faut en fait retenir de cette dernière série de modifications que lorsque l'on utilise des boucles, il faut, en fonction du problème que l'on veut résoudre, choisir avec une très grande minutie les valeurs des bornes des boucles.

Le programme modifié s'écrit :

```
 5  REM RECHERCHE DES CARRES DE PYTHAGORE
    JUSQU'A UN CERTAIN RANG
10  CLS
15  INPUT "VEUILLEZ ENTRER LA BORNE SUPERIEURE
    N" ; N
20  FOR C=1 TO N
25  C%=INT(C)
30  C1%=C%*C%
35  N1=C-1
40  N2=INT (SQR(2*C-1))
45:     FOR B=1 TO N1
50:     B%=INT(B)
52:     R%=C1%-B%*B%
55:     N3=B-1
60:         FOR A=N2 TO N3
65:         A%=INT(A)
70:         IF(R%-A%*A%=0) THEN PRINT A, B, C
75:         NEXT A
80:     NEXT B
85  NEXT C
90  END
```

- Remarquez l'utilisation du caractère : qui permet d'améliorer la lisibilité.

3.6.2 REPEAT ... UNTIL

Voyons un exemple de cet autre type de boucle

```
 10 REM DEVINETTE D'UN CHIFFRE CHOISI PAR ORIC
 20 NB%=INT(9*RND(1))
 30 REPEAT
 40 INPUT "ENTREZ UN CHIFFRE";C%
 50 NC%=NC%+1
 60 UNTIL (NB%=C%)
 70 PRINT "VOUS AVEZ TROUVE EN";NC%;" COUPS"
 80 ON NC% GOTO 90, 90, 90, 110, 110, 110,
    130, 130, 130
 90 PRINT "CHANCE EXCELLENTE"
100 END
110 PRINT "CHANCE MOYENNE"
120 END
130 PRINT "PAS DE CHANCE"
140 END
```

Ce qui peut se traduire littéralement par :

"Répéter les instructions situées entre REPEAT et UNTIL jusqu'à ce que la réponse au test soit positive".

La fonction de cette boucle est donc un peu différente de FOR ... TO ... NEXT ... car c'est à vous de prévoir les itérations successives et les incrémentations sur la variable testée. Cette possibilité d'incrémenter à l'aide d'une fonction quelconque apporte plus de souplesse, mais il se peut que l'on ne puisse sortir de la boucle.

Exemple :

```
 5 REPEAT
10 PRINT "JE BOUCLE SANS FIN, SANS FIN..."
15 UNTIL (I=1)
```

Vous voyez dans ce cas extrême que si vous n'incrémentez pas suffisamment sur la "variable de boucle" qui est testée, la boucle REPEAT ... UNTIL est sans fin (par analogie avec la boucle FOR ... NEXT, on appelle variable de boucle la variable sur laquelle s'effectue le test).

Pour reprendre la main faire [CTRL] [C] ou [RESET].

3.7 SOUS-PROGRAMMES

3.7.1 Présentation

La notion de sous-programme n'est pas évidente a priori. Présentons-la dans un exemple.

```
 10 X=0 : Y=0
 20 GOSUB 70
 30 PRINT A
 40 X=5 : Y=15
 50 GOSUB 70
 55 PRINT A
 60 END
 70 REM SOUS-PROGRAMME
 80 IF (X=5) THEN PRINT X ELSE A=Y-X
 90 IF (X=0) THEN PRINT X ELSE A=X-Y
100 RETURN
```

Un sous-programme est utile pour éviter de répéter plusieurs fois des lignes d'instructions identiques, dès que l'on utilise deux fois une même séquence basic.

La structure d'un programme utilisant un sous-programme est la suivante :

```
      :                    1000
150 GOSUB 1000              :
160 :                      2000 RETURN
      :
250 GOSUB 1000
260 :
```

L'utilisation d'un sous-programme permet aussi de décomposer un programme en parties simples à mettre au point.

Sur cet exemple nous voyons que l'ordre GOSUB sert à appeler le sous-programme. L'entier qui se trouve derrière ce mot est le numéro de ligne de la première instruction du sous-programme. Cet entier doit être explicité et non calculé. Cet entier doit bien entendu correspondre à un numéro de ligne existant, sinon Oric affichera le message d'erreur UNDEF'D STATEMENT IN (numéro de ligne où figure le GOSUB erroné). Le mot RETURN signifie au basic que le sous-programme est terminé. Donc Oric ne reconnaît qu'une partie de programme est un sous-programme que s'il y a un GOSUB indiquant le numéro de ligne où commence le sous-programme, et un RETURN indiquant sa fin. Les capacités de la machine sont telles que l'on ne peut pas appeler plus de vingt-cinq sous-programmes en même temps (ceci se réalise en construisant des sous-programmes qui font appel à d'autres sous-programmes). Si on le tente il y a affichage du message :

```
OUT OF MEMORY
```

C'est pourquoi le programme suivant :

```
  1 GOSUB 100
  5 END
100 I=I+1
110 GOSUB 100
120 RETURN
```

est interdit puisque le sous-programme écrit sur les lignes 100, 110 et 120 s'appelle lui-même.

Dans ce cas précis, la machine considèrera, après la 25ème "itération" (appel du sous-programme par le sous-programme) qu'il y a plus de 25 appels à des sous-programmes en même temps, et elle affichera donc le message d'erreur ci-dessus.

Pour la cohérence des programmes, il est vivement conseillé d'entrer dans un sous-programme par un GOSUB et d'en sortir par l'ordre RETURN ; et non d'utiliser l'ordre GOTO.

3.7.2 ON ... GOSUB

Cet ordre est similaire à l'ordre ON ... GOTO, dans une certaine mesure, ainsi que nous allons nous en rendre compte avec l'exemple qui suit :

```
  5 FOR I=1 TO 2
 10 ON I GOSUB 100,200 : NEXT I
 15 END
100 PRINT "LORSQUE I=1 LE SOUS-PROGRAMME DE
    LA LIGNE 100 EST APPELE"
110 RETURN
200 PRINT "LORSQUE I=2 LE SOUS-PROGRAMME DE
    LA LIGNE 200 EST APPELE"
210 RETURN
```

Vous voyez donc que ON ... GOSUB fait la même chose que ON ... GOTO, à ceci près que GOTO provoque un transfert inconditionnel alors que GOSUB est un appel à un sous-programme. Nous vous rappelons donc les règles de syntaxe à respecter :

I, qui correspond à la position des étiquettes derrière GOSUB, doit être entier (en fait non alphanumérique), et compris entre 0 et 255.

Si I est nul, ou si I est plus grand que le nombre de numéros de ligne situés derrière GOSUB, alors l'ordre ON ... GOSUB est ignoré (ne provoquera pas d'appel à un sous-programme). Bien entendu, les numéros de ligne sont entiers et explicités, ils correspondent à des numéros de ligne existant dans le programme.

3.7.3 CALL

Exemple : CALL A

Cet ordre est un appel à un sous-programme écrit en code machine (aussi appelé langage machine).

A est un entier en base hexadécimale qui indique l'adresse, en mémoire de la machine, de la première instruction du sous-programme.

L'utilisation du langage machine demande beaucoup de précaution dans la mesure où l'interpréteur basic de votre Oric ne peut pas en contrôler la validité.

De plus, vous devez être absolument certain que les instructions que vous allez utiliser ne vont pas "polluer" les zones mémoires réservées au basic de l'Oric. Dans le cas où vous modifiez le contenu de ces mémoires, il se pourrait que vous ne puissiez même pas revenir au programme basic qui appelait le sous-programme en langage machine.

3.7.4 DEF USR

Exemple : DEF USR=A avec A entier en base hexadécimale.

Cette instruction permet de définir l'adresse en mémoire de la première ligne d'un programme écrit en langage machine.

3.8 LES FONCTIONS

3.8.1 Fonctions numériques internes

Plusieurs fonctions numériques existent dans le basic de l'Oric. Ces fonctions sont monovariables, c'est-à-dire qu'elles n'agissent que sur **UNE** seule variable.

Nous donnons ci-dessous une liste de ces fonctions sans les expliquer en détail, car elles ont été déjà utilisées précédemment, ou bien elles demandent d'importantes connaissances mathématiques.

- GENERATION D'UN NOMBRE ALEATOIRE

RND(X) retourne un nombre aléatoire compris entre 0 et 1.

si X est positif, un nouveau nombre est généré à chaque appel,

si X est nul, le dernier nombre est généré à nouveau,

si X est négatif, le nombre correspondant à X est généré (table de nombres en mémoire).

- FONCTIONS MATHEMATIQUES SIMPLES

ABS(X) valeur absolue de X

INT(X) partie entière de X (plus grand entier inférieur à X

SGN(X) signe de X

SQR(X) racine carrée de X

- FONCTIONS MATHEMATIQUES COMPLEXES

ATN(X) arctangente de X

COS(X) cosinus de X

EXP(X) exponentielle de X

LN(X) logarithme népérien de X

LOG(X) logarithme décimal de X

SIN(X) sinus de X

TAN(X) tangente de X

Ces fonctions sont décrites avec plus de détails en annexe A.

3.8.2 Fonctions traitant les chaînes de caractères

Oric offre plusieurs fonctions à une seule variable permettant de manipuler les chaînes de caractères. Celles-ci peuvent se classer en trois groupes :

- conversion code ASCII/caractère et réciproque
- conversion chiffre/caractère et réciproque
- manipulation, extraction de chaînes de caractères

- FONCTIONS DE CONVERSION CODE ASCII/CARACTERE

Ces fonctions sont utiles pour avoir accès à des caractères spéciaux, ou pour accéder directement à la représentation des chaînes de caractères en mémoire.

CHR$(X) génère le caractère de code ASCII égal à X

$0 \leqslant X \leqslant 31$	caractères spéciaux
$32 \leqslant X \leqslant 128$	lettres et chiffres

ASC(X$) fournit le code ASCII du premier caractère de la chaîne X$.

Exemple : 10 PRINT CHR$(65) donne le caractère A
10 PRINT ASC("A") donne la valeur 65

- FONCTIONS DE CONVERSION CHAINE DE CARACTERES/NOMBRE

Ces fonctions sont utiles si l'on veut par exemple demander dans un programme l'entrée d'une donnée qui peut être une lettre (A-Z), ou un chiffre (0-9).

STR$(X) crée une chaîne contenant les chiffres X

Exemple : 10 A$=STR$(45)
Après exécution de cette instruction, la variable alphanumérique A$ contiendra les caractères "45".

VAL(X$) est l'entier correspondant à la valeur numérique de la chaîne

Exemple : 10 B$="1579ABC"
20 C=VAL(B$)
Après exécution des instructions 10 et 20, la variable numérique contiendra la valeur 1579.

- FONCTIONS DE TRAITEMENT DE CHAINES DE CARACTERES

Plusieurs fonctions du basic de l'Oric permettent d'extraire des caractères ou des sous-chaînes d'une grande chaîne.

LEFT$(A$,X) crée la chaîne constituée des X premiers caractères de A$.

Exemple : 10 C$=LEFT$("ABCDEFGHIJ",3)
La variable alphanumérique C$ contiendra ABC.

La fonction RIGHT$ joue le même rôle que LEFT$, mais en extrayant les caractères à partir de la droite.

Exemple : RIGHT$("0123456",4)="3456"

Il est possible d'extraire des caractères situés au milieu d'une chaîne de caractères.

La fonction MID$(A$,D,X) a un argument supplémentaire qui est le rang, dans la chaîne, du caractère à partir duquel on désire effectuer l'extraction.

Exemple : MID$("0123456",3,4)="2345"

Deux autres fonctions encore existent, qui traitent les chaînes de caractères.

LEN(A$) fournit la longueur de la chaîne A$

Exemple : LEN("0123456")=7

+ est un opérateur de concaténation (mise bout à bout) de chaînes de caractères.

Exemple : "01"+"23456"="0123456"

3.8.3 Création de fonctions utilisateurs

Supposez que dans un programme vous ayez besoin - à de nombreuses reprises et sans possibilité d'utiliser des boucles - d'une fonction du type :

$$\sin(5x)+3x^2\exp(x+3)+ABS(1/x+\cos x)$$

Vous vous rendez bien compte qu'il serait vite fastidieux de réécrire à chaque fois cette expression numérique. C'est pourquoi votre Oric dispose d'instructions permettant de construire vos fonctions et de vous en servir comme de fonctions internes de la machine.

Ces instructions sont :

DEF FN ... pour la construction de la fonction

FN ... pour son appel

Exemple d'utilisation :

- construction de la fonction "explicite" :

```
1000 DEF FNA(X)=SIN(5*X)+3*X*X*EXP(X+3)+ABS((1/X)+COS(X))
```

La syntaxe est donc DEF FN nom de la fonction = expression numérique ou alphanumérique. Le nom de la fonction suit les règles de formation des noms de variables, c'est-à-dire :

. le premier caractère du nom est alphabétique (A, ..., Z majuscule) ;

. les autres caractères du nom sont alphanumériques (A, ..., Z, 0, ..., 9) ;

. seuls les deux premiers caractères sont pris en compte par la machine pour caractériser la fonction ;

. pas de mots réservés dans les noms de fonctions.

Les fonctions peuvent être numériques ou alphanumériques. Ainsi :

DEF FNA(X) définit une fonction numérique (réelle).

DEF FNA$(X) définit une fonction alphanumérique.

Bien entendu l'expression derrière le signe = doit être du même type que la fonction.

Ce sont les caractères FN placés devant le nom de la fonction qui indiquent à l'interpréteur basic qu'il n'est pas en présence d'une variable indicée.

Les "appels" à la fonction utilisateur s'effectuent de la façon suivante.

Sur l'exemple précédent :

```
2005 PRINT FNA(5)
```

En fait vous devez faire comme si (FN nom de la fonction) était le nom de votre fonction et vous en servir comme d'une fonction interne à la machine. De plus sa définition doit précéder son utilisation.

Bien entendu ces fonctions dites explicites sont, ainsi que les fonctions internes, dépendantes d'une seule variable. Si vous avez besoin de fonctions agissant sur plusieurs variables vous ferez appel à ce qui a été vu au paragraphe précédent : les "sous-programmes".

3.9 LES ORDRES D'ENTREES/SORTIES

Les instructions que nous allons voir permettent d'échanger des informations entre l'extérieur (c'est-à-dire vous) et le microordinateur. Certaines de celles-ci vous sont déjà familières.

3.9.1 INPUT

Exemple d'utilisation :

```
5 INPUT "ENTREZ LES DONNEES",I1;I,,I$
RUN
```

Lorsque l'interpréteur basic rencontre cet ordre, il écrit le commentaire suivant :

```
ENTREZ LES DONNEES ?
```

et il attend que vous entriez les données (tant qu'elles ne sont pas entrées, le programme est stoppé). Pour entrer les données vous pouvez faire ainsi :

```
1.23,4.50 E3,"TOTO" [RETURN]
```

les trois données sur la même ligne séparées par une ",".

```
Ou bien     ?  1.23      [RETURN]
            ?? 4.50 E3   [RETURN]
            ?? "TOTO"    [RETURN]
```

et toutes solutions intermédiaires entre ces deux extrêmes.

Les données ne sont prises en compte que si vous tapez sur [RETURN], donc toute donnée peut être corrigée avant d'actionner cette touche. Par contre les données ne

sont pas conservées d'une exécution sur l'autre, ce qui oblige à les retaper toutes à chaque exécution. Bien entendu, vous devez entrer autant de données qu'il y a de variables derrière INPUT, et les données doivent être du même type que les variables auxquelles elles sont affectées.

3.9.2 GET

Exemple d'utilisation :

```
 5 GET A
10 PRINT A
RUN
```

Lorsque vous utilisez cette instruction, le basic attend une donnée constituée d'un seul caractère, qui est soit numérique (0-9) si la variable est réelle, ou quelconque si la variable est alphanumérique (le caractère est tout de même différent de "). L'exécution est arrêtée tant que le caractère n'est pas tapé sur le clavier. Le basic prend en compte le caractère dès que vous le tapez sur le clavier (pas besoin de [RETURN] pour enregistrer la donnée). Le caractère tapé n'apparaît pas sur l'écran (aucun moyen de correction possible par l'opérateur. Correction par le programme).

3.9.3 KEY$

Exemple d'utilisation :

```
 5 REPEAT
10 PRINT "TAPEZ SUR + POUR ARRETER"
15 UNTIL (KEY$="+")
20 END
```

KEY$ effectue grosso modo la même chose que GET, dans la mesure où on rentre un caractère du clavier ; mais la différence fondamentale est que le programme n'est pas stoppé si vous ne répondez pas (si vous n'entrez pas de données). Cet ordre est donc essentiel pour les jeux (graphiques en particulier), qui seront abordés au prochain chapitre.

3.9.4 READ - DATA

Exemple d'utilisation :

```
 5 READ A,B,C$
20 PRINT A,B,C$
50 DATA 1.253,454,"TOTO"
60 END
```

READ assigne aux variables A, B, C$ des valeurs stockées pour une instruction DATA. READ joue à peu près le même rôle que INPUT, sinon qu'au lieu de rentrer les données en cours d'exécution, elles sont entrées dans le programme grâce à des instructions DATA. Les caractères séparant les données dans les instructions DATA sont des virgules. La syntaxe est :

READ variable 1, variable 2, ...

Si variable 1 est la $n^{ième}$ variable rencontrée depuis le début du programme dans un ordre READ, elle recevra la valeur de la $n^{ième}$ donnée stockée dans un ordre DATA ou dans des ordres successifs. Bien entendu la valeur de la $n^{ième}$ donnée et $n^{ième}$ variable doivent être de même type. Cette assignation $n^{ième}$ variable - $n^{ième}$ donnée est rendue possible grâce à un "pointeur de liste de données" qui passe d'une donnée à une autre au fur et à mesure de leur lecture.

3.9.5 RESTORE

Cet ordre permet de remettre le "pointeur de données" à sa position initiale.

Exemple d'utilisation :

```
 5 READ A,B,A$
10 RESTORE
15 READ A1,B1,A1$
20 PRINT A,B,A$,A1,B1,A1$
25 DATA 1,0,"LE CHAPITRE EST FINI"
30 END
```

3.10 CONCLUSIONS

Au cours de ce long chapitre nous avons vu les ordres basic qui constitueront la charpente de vos programmes.

Nous avons essayé de vous sensibiliser à certains aspects de la programmation et à certaines méthodes permettant d'améliorer vos programmes. Pour bien saisir toutes les possibilités de ces instructions, vous devrez vous armer d'un peu de patience et surtout vous entraîner à faire vos propres programmes.

Dans les prochains chapitres vous allez apprendre à vous servir d'instructions plus distrayantes, permettant d'utiliser les possibilités graphiques et sonores de votre Oric.

CHAPITRE **4**

Possibilités graphiques de l'oric

4.1 INTRODUCTION

Au moment où vous entamez la lecture de ce chapitre, vous avez déjà amplement fait connaissance avec votre microordinateur. Vous avez, en effet, appris à jouer et à converser avec votre machine ; vous avez également commencé à programmer, en utilisant le langage BASIC, ce qui vous permet d'ores et déjà de concevoir des programmes relativement élaborés.

Pourtant il vous reste encore à découvrir toute la gamme de possibilités mise à votre disposition par Oric, et à compléter l'ensemble des commandes basic rencontrées jusqu'à présent.

Votre microordinateur va en effet vous permettre de réaliser des graphiques simples, complexes, colorés, immobiles, mouvants, ou encore clignotants si vous le désirez. Il vous permettra également de construire des sons, ou même des morceaux de musique si vous êtes passionné de solfège.

Inutile de préciser le parti que vous pouvez tirer de ces capacités sonores et graphiques pour améliorer la richesse et la variété de vos jeux, ou simplement agrémenter vos programmes par des couleurs ou des sons.

Nous commencerons par présenter les possibilités graphiques en définissant tout d'abord les divers modes graphiques que vous utiliserez, puis en passant en revue les commandes de chaque mode.

4.2 LES MODES GRAPHIQUES

On peut classer ces modes en deux groupes :

Le premier est constitué par les modes graphiques basse résolution.

le second groupe comprend le mode unique correspondant aux graphiques haute résolution.

Nous allons préciser, dans les paragraphes suivants, les notions de mode, et de haute ou basse résolution.

4.2.1 Les modes basse résolution

TEXT, LORES 0, LORES 1, sont les noms des trois modes graphiques composant le premier groupe. Vous connaissez très bien le premier d'entre eux : TEXT, puisque c'est celui que vous avez utilisé jusque là.

Qu'est-ce qu'un mode graphique ?

Un mode graphique, avec l'ensemble des commandes qu'il comprend, constitue une option qui conditionne la manière d'écrire ou de faire imprimer un point ou un caractère sur l'écran.

Décrivons le mode TEXT pour préciser davantage. Lorsque vous mettez Oric en marche, c'est ce mode qui apparaît automatiquement. Comme vous l'avez remarqué, vous disposez, en mode TEXT, de la totalité de l'écran pour écrire vos programmes ou lancer vos ordres, hormis la première colonne sur laquelle vous ne pouvez pas écrire. Lorsque toutes les vingt-sept lignes de l'écran ont été utilisées, celles-ci défilent vers le haut, jusqu'à laisser apparaître une ligne libre en bas de l'écran.

Jusqu'à présent, vous n'avez utilisé le mode TEXT que

pour des besoins d'écriture. Vous allez maintenant apprendre à vous en servir pour réaliser vos premiers dessins, grâce aux commandes qui seront détaillées dans les paragraphes suivants.

Mais auparavant, présentons les deux autres modes graphiques basse résolution : LORES 0 et LORES 1.

On passe du mode TEXT au mode LORES 0 ou LORES 1 en tapant LORES 0 ou LORES 1 ; on repasse au mode TEXT en tapant TEXT. Le nom du mode peut donc être considéré comme une commande permettant d'accéder à ce mode, commande qu'on peut utiliser dans un programme.

Après avoir tapé : LORES 0, vous voyez l'écran se transformer. Vous constatez alors que le fond de l'écran devient noir à l'exception de la ligne du bas et des deux premières colonnes. Le curseur se trouve dans cette dernière ligne. Si vous y écrivez, vous verrez la surface en fond noir diminuer au fur et à mesure que vous introduisez de nouvelles lignes. Les premières colonnes à gauche sont inaccessibles à l'écriture.

Vous avez peut-être déjà tapé LORES 1. Là encore, l'écran se transforme de la même façon que pour LORES 0. Mais quoi que vous essayiez de taper, vous verrez apparaître à la place des caractères attendus (et habituels) des blocs aux formes étranges. Vous venez de prendre contact avec la "forme alternée des caractères", et en même temps de mettre en évidence la seule différence entre les modes LORES 0 et LORES 1.

Vous n'aviez jusqu'ici utilisé que la "forme standard" des caractères, à laquelle nous sommes tous habitués ; c'est celle que nous rencontrons en mode TEXT ou en mode LORES 0.

Mais quelle que soit la représentation choisie : forme standard en mode TEXT ou LORES 0, ou forme alternée en mode LORES 1, les caractères ont toujours la même signification. Ainsi, l'ordre "PRINT", commun aux trois modes, exerce la même action en mode LORES 0 ou en mode LORES 1.

En mode TEXT, la première colonne de gauche abrite un

paramètre contrôlant la couleur du fond de l'écran ; la seconde abrite un paramètre contrôlant la couleur de l'avant-plan, des caractères. En mode LORES (0 ou 1) apparaît un paramètre supplémentaire - contrôlant les formes standard ou alternées des caractères - qui est placé dans la première colonne de gauche.

Il y a près d'une centaine de caractères (A, , , $, etc...). A chacun d'eux on peut associer une forme étrange, qui est la forme alternée du caractère. Le nombre et la variété de ces nouveaux caractères vous permettront de construire des dessins originaux en associant les blocs les uns aux autres.

Comment obtenir la forme alternée d'un caractère standard, d'une manière simple et rapide ? L'instruction suivante vous le montre.

```
PRINT 65, CHR$(27);"I";CHR$(65)
```

Vous verrez alors s'imprimer 65 et plus loin un bloc correspondant au caractère alterné de A. En effet, 65 représente le nombre code ASCII du caractère A. (ASCII pour American Standard Code for Information Interchange). Vous trouverez en annexe la liste des codes ASCII de tous les caractères ; ce sont des nombres compris entre 32 et 125. La commande permettant d'obtenir le caractère correspondant à son nombre code est CHR$; ainsi CHR$(65)=A.

Vérifiez-le en tapant :

```
PRINT CHR$(65)
```

La commande réciproque permettant d'obtenir le code ASCII d'un caractère est ASC :

```
ASC("A")=65
```

Mais 27 est inférieur à 32 : il ne représente donc pas un nombre code ASCII. Ici, 27 est un attribut, c'est-à-dire qu'on peut lui faire jouer un rôle particulier. Par exemple ici l'ensemble CHR$(27);"I" a pour action de sélectionner la forme alternée de CHR$(65). Si vous supprimez CHR$(27);"I"; et tapez : PRINT 65, CHR$(65), vous verrez A s'imprimer après 65.

Vous pouvez bien sûr remplacer 65 par tout autre nombre compris entre 32 et 125, mais vous devrez garder CHR$(27);"I".

Exemple : PRINT 100, CHR$(27);"I";CHR$(100)

27 n'est pas le seul attribut pouvant être utilisé après CHR$. Les nombres entre 0 et 31 peuvent être considérés comme attributs.

Employés avec CHR$, ils ont surtout un intérêt pour les sorties sur imprimante. Ils permettent entre autre de choisir la grandeur des lettres, ou de faire avancer le papier, etc... Certains ont cependant une action sur les sorties écran ; tapez :

```
PRINT CHR$(12)
```

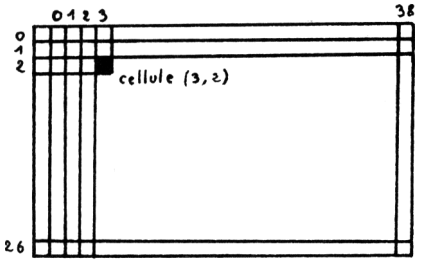
Vous obtiendrez un résultat équivalent à celui de l'ordre CLS.

Nous utiliserons par la suite CHR$ avec d'autres attributs en particulier 8 et 9.

Nous n'avons fait, pour l'instant, que présenter les trois modes graphiques basse résolution. Il est temps d'entrer plus avant dans les détails.

Commençons par expliciter le terme basse résolution.

En mode LORES 0, et LORES 1 ou TEXT, l'écran est divisé en un certain nombre d'unités élémentaires : des cellules, qui constituent un quadrillage de cet écran. Ces cellules sont de petits carrés : il y en a 39 colonnes verticales, et 27 lignes horizontales.



Ces cellules, comme on peut le voir sur le schéma, sont numérotées de 0 à 38 dans le sens horizontal et de 0 à 26 dans le sens vertical.

En fait, ce quadrillage peut servir à constituer un repère formé d'une origine et de deux axes. L'origine sera, par définition, le coin en haut et à gauche de l'écran, les deux axes seront x et y définis par le schéma :

x horizontal de gauche à droite ;
y vertical de haut en bas.

La cellule de coordonnées x et y dans ce repère est donc repérée par le couple (x,y) comme l'indique l'exemple sur le schéma. $0 \leqslant x \leqslant 38$, $0 \leqslant y \leqslant 26$, x et y sont comptés à partir de l'origine.

La taille des cellules est telle qu'un caractère, quel qu'il soit, occupe exactement une cellule.

Cette division de l'écran constitue ce que l'on appelle la basse résolution. En effet il est possible de diviser l'écran beaucoup plus finement encore : c'est l'objet du mode haute résolution.

4.2.2 Le mode haute résolution

Il se nomme HIRES ; son nom, comme celui des autres modes, constitue une commande qui permet d'accéder à ce mode.

Ayant tapé HIRES vous voyez alors le fond de l'écran devenir noir, mais constatez aussi qu'il reste au bas de l'écran trois lignes de mode TEXT. Ces trois lignes constituent une fenêtre qui vous permet de formuler des commandes (et surtout de voir ce que vous écrivez), tout en ayant sous les yeux les dessins que vous êtes en train de tracer.

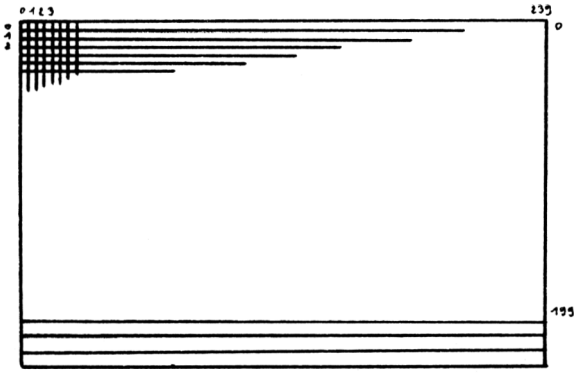
Nous sommes en haute résolution : la division de l'écran en unités élémentaires est donc beaucoup plus fine qu'en basse résolution. Les cellules des modes TEXT et LORES sont devenues des pixels, c'est-à-dire des points, constituant l'élément graphique de base.

Tapez : CURSET 120,100,1 (en mode HIRES)

Vous verrez apparaître un point au centre de l'écran qui vous montrera ce qu'est un pixel. (Nous reviendrons bien sûr sur la commande CURSET tout-à-l'heure).

Horizontalement, l'écran est divisé en 240 pixels, tandis que verticalement, il est divisé en 200 pixels. En mode HIRES, tout l'écran est disponible en largeur pour l'impression des pixels.

On peut là encore élaborer un repère à partir du maillage constitué par les pixels. L'origine est toujours le coin haut à gauche de l'écran, un axe x horizontal va de gauche à droite, un axe y vertical va de haut en bas.



Lignes en mode TEXT

Les pixels sont numérotés de 0 à 239 horizontalement, et de 0 à 199 verticalement.

Voici donc présenté ce mode haute résolution graphique.

Si vous cherchez à comparer les divers modes graphiques, vous constaterez que TEXT et LORES 0 sont très proches l'un de l'autre. Nous avons vu ce qui différenciait LORES 1 de LORES 0.

Quant au mode HIRES, nous pouvons dire qu'il n'a rien de commun avec les trois premiers, si ce n'est les trois lignes de mode TEXT. Nous verrons qu'également, au niveau des commandes, LORES 0, LORES 1 et TEXT sont assez voisins l'un de l'autre (les commandes diffèrent peu ou pas d'un mode à l'autre).

Par contre le mode HIRES a des commandes qui lui sont spécifiques.

Les premières commandes graphiques que l'on va introduire sont des commandes communes aux quatre modes TEXT, LORES 0, LORES 1 et HIRES. Elles concernent les couleurs.

On distingue les couleurs d'arrière-plan ou "papier" et les couleurs d'avant-plan : l'"encre". Il y a donc deux commandes qui sont :

PAPER N

et INK N

où N est un nombre variable compris entre 0 et 7, qui contrôle la couleur, suivant le tableau suivant :

0 noir
1 rouge
2 vert
3 jaune
4 bleu
5 nagenta
6 cyan
7 blanc

Essayez ces deux commandes, jusqu'à trouver les couleurs du fond d'écran, et les caractères qui vous conviennent le mieux. Bien sûr, si l'encre et le papier sont de même couleur vous ne voyez plus rien.

Tapez :

```
10 FOR A = 0 TO 7
20 FOR B = 7 TO 0 STEP -1
30 INK A : PAPER B
40 WAIT 100
50 NEXT B
60 NEXT A
```

4.3 LES COMMANDES GRAPHIQUES

4.3.1 Commandes graphiques basse résolution

La première commande rencontrée sera PLOT X,Y,A$. Utilisons-la tout de suite.

Vous êtes en mode TEXT ou LORES 0 (ou LORES 1)

Tapez :

```
PLOT 15,10,"BONJOUR"
```

En mode TEXT et LORES 0, BONJOUR va aller s'imprimer au milieu de l'écran, le caractère B occupant la cellule (15,10). En LORES 1, le résultat sera le même, mais BOUJOUR (comme d'ailleurs votre commande) sera écrit sous forme alternée.

PLOT x,y,A$ placera donc le premier caractère de la chaîne A$ dans la cellule de coordonnées x, y. Attention à ne pas sortir de l'écran en donnant un x ou un y trop grand. Si vous tapez :

```
PLOT 10,10,"H"
```

vous verrez apparaître un H sur l'écran. A la place d'une chaîne alphanumérique, on peut aussi mettre un nombre, pour le troisième argument de PLOT. Ainsi, tapez :

```
H=65
PLOT 15,10,H
```

La caractère A apparaît, car 65 est le code ASCII de A.

Voici un programme utilisant les formes alternées des caractères et la commande PLOT. Il s'agit d'un balayage horizontal de l'écran par des caractères de forme alternée. Ceci montre l'importance de la commande PLOT, qui vous permet de placer facilement un caractère et de le faire évoluer sur l'écran, ce qui peut avoir un grand intérêt pour les jeux.

```
  1 REM *** BALAYAGE ***
  5 LORES 1
  6 A=0:C$=" "
 10 A$="6I"
 20 REPEAT
 30 FOR B=0 TO 35
 40 PLOT B,A,A$
 50 PLOT B,A,C$
 60 NEXT B
 70 A=A+1
 80 FOR C=35 TO 0 STEP-1
 90 PLOT C,A,A$
100 PLOT C,A,C$
110 NEXT C
120 A=A+1
130 UNTIL A=26
```

Ce programme vous montre comment créer l'illusion du mouvement : voir les lignes 40 et 50, ou les lignes 90 et 100.

Si vous voulez avoir les caractères standard, remplacez en ligne 5 LORES 1 par LORES 0.

Si vous êtes exigeant au point de vouloir mélanger les deux types de caractères, vous en avez la possibilité. Voyons comment.

On utilise deux expressions qui ont une action réciproque l'une sur l'autre. L'expression CHR$(8) permet, lorsqu'on est en mode LORES 1, de faire apparaître des caractères standard.

L'expression CHR$(9) permet, réciproquement, lorsqu'on se trouve en mode LORES 0, d'obtenir des caractères de type alterné. De plus, CHR$(9) peut servir, en LORES 1, à revenir aux caractères de type alterné, après utilisation de CHR$(8).

De même, CHR$(8) sert, en LORES 0, à retrouver les caractères de type standard après utilisation de CHR$(9). (Souvenez-vous que pour $x \leqslant 31$, CHR$(x) exerce une action particulière, et ne donne un caractère que pour $x \geqslant 32$, x étant le nombre code ASCII de ce caractère).

Comment utilise-t-on CHR$(8) et CHR$(9) dans la pratique ?

Les manipulations suivantes fournissent la réponse.

Tapez :

```
10 LORES 1
20 A$=CHR$(8)+"BONJOUR"+CHR$(9)
30 B$="LORES 1"
40 PLOT 15,10,B$
50 PLOT 15,12,A$
```

puis :

```
10 LORES 0
20 A$=CHR$(9)+"BONJOUR"+CHR$(8)
30 B$="LORES 0"
40 PLOT 15,10,B$
50 PLOT 15,12,A$
```

La concaténation (opération rencontrée aux lignes 20), permet, lorsque l'on est en LORES 1 par exemple, de considérer à l'exécution de la commande PLOT 15,12,A$, la chaîne de caractères : BONJOUR, comme filtrée par CHR$(8) ce qui va la faire imprimer en caractères standard. L'impression de BONJOUR une fois réalisée, CHR$(9) fait revenir à la forme originale.

La seconde commande est SCRN(x,y). SCRN vient de SCREEN (écran en anglais). Son but est d'aider l'utilisateur à connaître le caractère qui occupe la cellule de coordonnées x, y sur l'écran.

Ainsi, après avoir tapé CLS pour nettoyer l'écran, si vous effectuez la commande PLOT 10,15,"H", vous verrez bien sûr apparaître la lettre H dans la cellule de coordonnées 10, 15. Tapez alors :

```
PRINT SCRN(10,15)
```

Le nombre 72 vous sera alors fourni ; c'est en effet le code ASCII du caractère H.

Peut-être vous demandez-vous comment on peut utiliser cette commande. L'exemple qui suit vous montre les services qu'elle peut rendre dans un jeu.

```
  1 TEXT
  5 C$=" "
 10 FOR N=1 TO 8
 12 M=N+8
 20 PLOT 10,M,"O"
 30 NEXT N
 40 C=2
 50 A=INT(RND(1)*26)
 60 REPEAT
 70 PLOT C,A,"X"
 80 WAIT 4
 90 PLOT C,A,C$
100 IF SCRN(C+1,A)=79 THEN C=34
110 C=C+1
120 UNTIL C=35
130 GOTO 40
```

A la ligne 100, 79 représente le code ASCII de O. On utilise alors le résultat numérique fourni par SCRN pour tester si au coup suivant le "X" atteindra le "O".

Terminons ce paragraphe sur les commandes basse résolution par l'utilisation des couleurs, et des attributs.

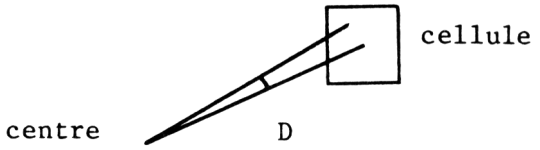
Tapez :

```
10 LORES 0 : F=0
20 C=0:A=10:B=10
30 R=10:D=2*PI/50
40 REPEAT
50 PLOT A+R*SIN(C),B+R*COS(C),18+F
60 C=C+D:IF C >2*PI THEN F=1
70 UNTIL C >4*PI
```

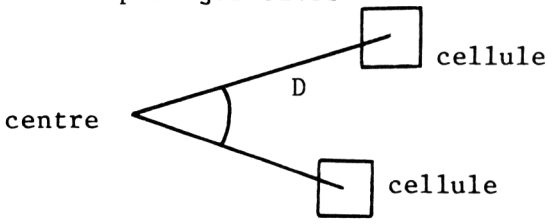
Commentaires :

La ligne 30 définit D comme l'incrément de l'angle dont on tourne pour placer les cellules les unes après les autres. La ligne 20 définit le centre du cercle par A et B, et le rayon par R. La ligne 50 choisit les couleurs des diverses cellules ; nous reviendrons sur ce point.

D doit être suffisamment grand pour qu'on puisse passer d'une cellule à la suivante assez rapidement.



Mais surtout, il ne doit pas être trop grand, sinon les cellules ne seront plus jointives.



D'autre part, x=A+R*sin(C) a peu de chances d'être entier. Comment la commande PLOT va-t-elle interprêter ces coordonnées non entières ? Elle va simplement choisir la cellule correspondant à INT(x),INT(y). C'est-à-dire que PLOT x,y,"A" est équivalent à PLOT INT(x),INT(y),"A". Vérifiez-le sur plusieurs exemples.

Parlons maintenant des couleurs et attributs. Vous le constatez : grâce à ce programme, on peut faire intervenir la couleur après la commande PLOT, par le choix des attributs de couleur. Les attributs considérés dans le programme contrôlent la couleur du papier (de la cellule). Ils sont alors compris entre 16 et 23, c'est-à-dire qu'ils valent 16+N, N variant de 0 à 7. Les attributs correspondant à l'encre varient de 0 à 7.

En mode LORES, la commande PLOT suivie d'un ordre de couleur, ne colorie qu'une cellule : tapez

```
LORES 0
PLOT 10,15,20
```

Si par les commandes INK ou PAPER vous changez l'encre ou le papier, vous ne changerez pas la couleur de cette cellule, et le fond - en LORES - restera toujours noir.

Mais en mode TEXT, PLOT suivi d'un attribut de couleur, colore la cellule choisie, mais également toute la ligne ou partie de ligne à droite de cette cellule.

Tapez :

```
TEXT : PAPER 7
PLOT 15,15,17
PLOT 20,15,23
```

Vous pouvez ainsi obtenir des surfaces quelconques colorées comme vous le voulez. Ici encore les ordres PAPER ou INK ne changent pas la couleur des cellules que vous avez colorées. Mais d'autres commandes PLOT sur ces cellules en changent bien sûr la couleur.

Les attributs de couleur sont donc les nombres de 0 à 7 pour l'encre, et de 16 à 23 pour le papier. Tous les attributs sont compris entre 0 et 31. Vous trouverez en annexe H leur liste et leur action. Mais montrons le rôle de l'attribut 12 tout de suite à titre d'exemple.

Tapez :

```
 1 TEXT
 5 FOR N=1 TO 24
10 PRINT "TEXTE POUR REMPLIR L'ECRAN"
20 NEXT N
30 PLOT 10,10,5
40 PLOT 11,11,18
50 PLOT 12,12,12
```

Vous constatez que l'attribut 12 produit un clignotement. Vous constatez également que les attributs n'affectent que les cellules situées à leur droite, et qu'une cellule est affectée par la commande PLOT + ATTRIBUT, située la première à sa gauche.

Enfin si vous tapez :

```
LORES 0
PLOT 10,10,19.1
```

Bien que 19.1 ne soit pas entier, la commande fonctionnera néanmoins ; là encore, elle équivaudra à PLOT 10,10,INT(19.1).

4.3.2 Commandes du mode HIRES

Pour obtenir ce mode, il suffit de taper : HIRES.

La première commande va nous faire découvrir le curseur HIRES. Cette commande est CURSET X,Y,FB. En tapant CURSET 120,100,1 vous verrez alors apparaître un point au centre de l'écran : c'est le curseur HIRES. X,Y sont les coordonnées du pixel, du point de l'image repérant le curseur. Pour la commande CURSET, elles seront toujours repérées par rapport au coin haut et gauche de l'écran.

Le troisième paramètre intervenant dans l'ordre CURSET est le nombre code FB (pour Foreground, Background). Il a quatre valeurs, c'est-à-dire :

FB = 0 couleur du fond (ou du papier des lignes TEXT de la fenêtre)
1 couleur de l'encre
2 inversion des couleurs
3 sans effet

FB conditionne la manière dont sera imprimé le curseur sur l'écran. Pour voir l'effet de ce nombre code, tapez :

```
HIRES
CURSET 120,100,1
CURSET 120,150,2
CURSET 120,50,3
CURSET 100,100,0
```

A cette commande CURSET, on peut associer une commande de même type : CURMOV X,Y,FB. La seule différence avec CURSET, c'est que pour CURMOV, X et Y sont les coordonnées relatives à la dernière position du curseur.

Tapez :
```
HIRES
CURSET 120,100,1
CURMOV 10,10,1
```

pour obtenir confirmation.

Attention à ne pas sortir de l'écran (Oric vous enverrait un message d'erreur) : si A et B sont les coordonnées de la dernière position du curseur, on doit respecter

$0 \leqslant X+A \leqslant 239$
$0 \leqslant Y+B \leqslant 199$

On peut bien sûr tracer des traits avec CURSET ou CURMOV mais cela oblige à placer le curseur successivement sur tous les points du trait.

Il existe une commande simple permettant de tracer un segment de droite : c'est l'ordre DRAW X,Y,FB. Il permet de tracer un trait à partir de la dernière position du curseur, jusqu'au point A+X, B+Y (si A et B étaient les coordonnées de la dernière position du curseur). DRAW déplace le curseur qui occupe désormais la position A+X, B+Y.

Exemple d'utilisation :

```
10 HIRES
20 CURSET 120,50,3
30 DRAW -50,50,1
40 DRAW 50,50,1
50 DRAW 50,-50,1
60 DRAW -50,-50,1
```

Améliorons encore nos possibilités en passant à la commande PATTERN.

PATTERN est un mot anglais signifiant modèle, ce qui va prendre tout son sens.

Nous venons d'utiliser la commande DRAW, en ne traçant que des traits continus. Vous voulez tracer facilement et rapidement des traits mixtes ou pointillés ? Rien de plus facile.

Avant tout, nous devons nous souvenir qu'Oric "pense" en binaire, c'est-à-dire uniquement avec des 0 ou des 1 groupés par huit, formant alors un octet.

Un octet peut donc représenter des nombres allant de:

00000000		11111111 (binaire)
	à	
0		255 (décimal)

C'est cette disposition de 0 ou de 1 à la suite les uns des autres qui permet d'obtenir des traits interrompus. Vous pouvez utiliser pour cela tout nombre entre 0 et 255 ; le trait obtenu sera interrompu selon la représentation binaire de ce nombre. Ainsi si vous tapez :

```
 1 HIRES
 5 CURSET 60,30,1
10 PATTERN 15
20 DRAW 120,0,1
```

15 en binaire s'écrit 00001111

Examinons le résultat obtenu. Vous constatez que les quatre premiers points du trait ne sont pas imprimés, que les quatre suivants le sont, et ainsi de suite.

Les quatre premiers points correspondent aux quatre 0 par lesquels commence le nombre 15 en binaire.

Les quatre suivants, vous l'avez compris, correspondent aux quatre 1 par lesquels finit le nombre 15 en binaire.

Si vous remplacez 15 par 170 - qui en binaire s'écrit 10101010, - vous obtiendrez des pointillés plus serrés.

Lorsque vous mettez Oric en marche, c'est le nombre modèle 255 qui est sélectionné ; c'est pourquoi vous tracez des traits pleins avec DRAW.

Pour le moment, en mode HIRES, vous n'avez pu faire imprimer des caractères que dans les trois lignes de mode TEXT du bas de l'écran. Or, il existe une commande qui va vous permettre de placer un caractère (ou plusieurs) sur la partie haute résolution de l'écran, à l'endroit qui vous plaît.

Cette commande est CHAR X,S,FB.

Vous êtes en mode HIRES. En tapant encore HIRES, vous enlevez de l'écran tous les dessins qui s'y trouvaient. Tapez alors CURSET 120,100,1 : vous voyez le curseur HIRES se placer au centre de l'écran. Tapez maintenant : CHAR 65,0,1 : vous voyez alors la lettre "A" s'imprimer au milieu de l'écran, là où se trouve le curseur HIRES.

Avant d'employer CHAR, vous devez donc choisir, par la commande CURSET, l'endroit de l'écran où vous désirez faire afficher un caractère. Puis, vous déterminez les trois paramètres X,A,FB qui suivent l'ordre CHAR.

X est le code ASCII du caractère que vous voulez faire imprimer ; ASC("A")=65.

S vaut 0 ou 1. Le 0 correspond à la forme standard des caractères, le 1 à la forme alternée.

FB est le nombre paramètre déjà rencontré dans les commandes CURSET, CURMOV ou DRAW, qui contrôle les couleurs.

Voici un programme, qui vous montre comment utiliser la commande CHAR.

```
 10 HIRES
 20 A$="BONJOUR"
 30 CURSET 30,30,3
 40 FOR N=1 TO 3
 50 READ X,Y
 60 FOR A=1 TO LEN(A$)
 70 CHAR ASC(MID$(A$,A,1),0,1)
 80 CURMOV X,Y,0
 90 NEXT A
100 NEXT N
110 DATA 10,10,0,10,10,-10
```

LEN est une fonction donnant la longueur d'une chaîne de caractères : LEN(A$).

MID$(A$,A,L) ne considère les caractères de la chaîne A$ qu'à partir du Aième et au nombre de L. Si A$= "BONJOUR" MID$(A$,3,2) équivaut à : NJ.

La commande que nous allons maintenant étudier va vous permettre de tracer des cercles très facilement.

Elle se présente comme suit : CIRCLE R,FB.

R est un paramètre correspondant au rayon du cercle.

FB est le nombre paramètre contrôlant les couleurs déjà rencontré. Pour tracer un cercle, vous devez connaître l'emplacement de son centre, et la valeur de son rayon.

Vous pouvez déterminer la position du centre par la commande CURSET.

Ainsi, si vous tapez :

```
HIRES
CURSET 120,100,3
CIRCLE 50,1
```

Vous obtiendrez un cercle de rayon 60, dont le centre a pour coordonnées 120 et 100. Attention à bien choisir la position du centre et la valeur du rayon, afin de ne pas sortir de l'écran.

Vos cercles sont tracés en trait plein. Mais comme pour la commande DRAW, vous pouvez faire intervenir la commande PATTERN pour obtenir des cercles en traits interrompus.

```
10 HIRES
20 CURSET 120,100,3
30 FOR X=49 TO 1 STEP-2
40 CIRCLE X,1
50 INK INT(X/7)
60 PATTERN 50-X
70 NEXT X
```

La commande suivante : POINT permet de déterminer si un point, un pixel de l'écran est imprimé dans la couleur de l'encre de l'arrière-plan, du papier, ou dans celle de l'avant plan.

Si vous êtes passé du mode HIRES en mode TEXT, vos dessins haute résolution ont bien sûr disparu, mais ils sont cependant conservés dans la mémoire de l'écran.

Pour les enlever de la mémoire, vous devez taper NEW. Revenons au mode HIRES. Tapez :

```
HIRES
CURSET 50,50,0
```

Tapez maintenant :

```
TEXT
PRINT POINT(50,50)
```

Oric vous répond 0 puisqu'en effet le curseur était imprimé dans la couleur du fond de l'écran (ou du papier). Si en mode HIRES, vous aviez tapé CURSET 50,50,1 c'est-à-dire si vous aviez fait imprimer le curseur dans la couleur de l'encre, alors une fois revenu en mode TEXT

```
PRINT POINT(50,50)
```

vous aurait fourni comme réponse : -1 (et non pas 1).

Passons maintenant à la commande la plus riche, sans doute la plus performante, mais aussi la plus complexe. Il s'agit de la commande FILL A,B,X.

Elle vous permet de remplir une portion de l'écran (déterminée par vous) dans la couleur que vous avez choisie. Ou encore, elle vous permet de faire clignoter les dessins se trouvant dans une certaine région de l'écran. Elle peut encore faire bien d'autres choses, et vous découvrirez l'ensemble de ses innombrables possibilités à l'usage.

FILL est suivi de trois paramètres A,B,X.

A correspond au nombre de lignes horizontales. En mode HIRES, il y a 200 lignes de pixels.

A est un paramètre qui contrôle le nombre de lignes sur lequel FILL va agir. De même B est un paramètre contrôlant le nombre de colonnes sur lequel va agir cette commande.

Quant à X, c'est le paramètre qui sert à obtenir la nature de l'effet cherché : couleur, clignotement, etc...

Voici un petit programme qui dispose des couleurs à l'aide de cette commande.

```
 5 HIRES
10 FOR N=1 TO 22
20 FILL 3,1,17
21 FILL 3,1,18
22 FILL 3,1,19
24 NEXT N
```

Montrons d'abord le rôle du paramètre A.

Tapez :
```
HIRES
FILL 10,1,17
FILL 30,1,18
```

Vous constatez deux choses :

10 lignes se remplissent de rouge à la première commande FILL ;
30 lignes se remplissent de vert à la seconde commande FILL.

A contrôle donc bien le nombre de lignes sur lesquelles va agir FILL.

Vous remarquez ensuite que le curseur HIRES se déplace lorsque vous donnez l'ordre FILL. En effet initialement, il était en (0,0) puis après le 1er ordre FILL, il est venu se positionner en (0,10), pour alors remplir les lignes 10 à 40 de vert.

Tapez :
```
CURSET 60,50,1
FILL 30,1,22
FILL 10,1,21
```

Cette fois, l'ordre FILL n'agit qu'à partir de la position du curseur qui vient se repositionner sur la même verticale lorsque l'ordre FILL est accompli, mais en

se décalant vers le bas du nombre de lignes modifiées par l'ordre FILL.

Cependant, si vous tapez :

```
CURSET 120,50,1
CURSET 123,50,1
FILL 10,1,17
```

vous constaterez que la commande FILL n'affecte l'écran qu'à partir de la première position du curseur. Cette remarque nous permet d'introduire le rôle du paramètre B.

Il faut considérer que, sur une ligne, il y a 240 points (ou pixels) rangés par blocs de 6 ; il y a donc 40 de ces blocs. La commande FILL n'agit pas sur un pixel isolé ; en fait elle exerce son action par blocs de 6 pixels.

Pour cela, si vous positionnez votre curseur au hasard sur l'écran, l'ordre FILL agira à partir de la coordonnée horizontale, multiple de 6, qui précède la position réelle du curseur. C'est ce que vous a montré le dernier exemple proposé.

Pour confirmation, tapez :

```
10 HIRES
20 CURSET 120,100,1
30 FOR N = 1 TO 25 STEP 1
40 FILL 1,1,18
50 CURSET 120-N, 100+N,1
60 NEXT N
```

Ainsi, au lieu d'apparaître sur une oblique, l'impression des couleurs va être disposée en escalier. Nous verrons tout à l'heure comment obtenir une oblique.

Tapez :

```
HIRES
FILL 10,1,63
FILL 10,2,63
FILL 10,5,63
```

Vous voyez apparaître, suivant l'ordre FILL considéré, 1, 2 ou 5 blocs (occupant 6 pixels en largeur). C'est bien sûr le paramètre B qui sert à changer le nombre de ces blocs.

Cependant, pour les couleurs, B ne joue aucun rôle.

```
HIRES
FILL 10,2,22
FILL 10,15,19
```

Toute la ligne se colore à partir du curseur.

Les deux premiers paramètres ont donc été présentés. Attention à ne pas choisir A trop grand (vous sortiriez de l'écran), et à ne pas prendre non plus des valeurs de B trop importantes : si vous êtes à l'extrême gauche de l'écran, B peut valoir au maximum 40 (il y a 240 = 6x40 pixels sur une ligne).

Passons maintenant au dernier paramètre : X.

Pour $0 \leqslant X \leqslant 127$, vous obtiendrez toute la gamme de possibilités mise à votre disposition par la commande FILL.

Entre 128 et 255 vous retrouverez les mêmes possibilités. Nous n'aurons besoin que de $X \leqslant 127$ pour ce paragraphe. En effet Oric, par la commande FILL, ne tient compte que des 6 premiers chiffres de X (en représentation binaire). Or de 128 à 255, il faut 7 bits (0 ou 1) pour représenter X en binaire.

Entre 0 et 31, X est un attribut.

De 0 à 7, il contrôle la couleur de l'encre :

```
HIRES
CURSET 120,100,1
CIRCLE 95,1
FILL 95,1,1
```

Vous voyez alors un quart du cercle devenir rouge.

De 8 à 15, X permet de passer aux caractères de forme alternée (s'il y a des caractères sur l'écran en partie haute résolution), ou bien de faire clignoter, ou encore d'imprimer les caractères en les doublant sur une verticale. Par exemple X = 12 fait clignoter.

Tapez :

```
HIRES
CURSET 120,100,0
CIRCLE 95,1
CURSET 0,0,0
FILL 198,1,12
```

De 16 à 23, X contrôle la couleur du papier :

```
HIRES
FILL 10,1,21
```

De 24 à 31, X dérègle la synchronisation du balayage de l'écran. Vous retrouverez l'action des attributs en annexe.

Si X ⩾ 32, X n'est plus un attribut, il devient un modèle (PATTERN) et ce jusqu'à 127.

Tapez :

```
HIRES
FILL 10,1,100
FILL 10,1,42
```

En binaire 100 s'écrit 1100100
42 s'écrit 0101010

FILL imprime les pixels correspondant aux 1 et non ceux correspondant aux 0. Mais comme le travail s'effectue par blocs de 6 pixels, seuls les 6 premiers bits à partir de la droite composant le nombre seront pris en compte. Ainsi pour 100, FILL lira 100100.

C'est précisément cette notion de modèle qui va nous permettre d'obtenir la disposition en oblique des régions affectées par la commande FILL.

```
HIRES
CURSET 120,100,1
FILL 1,1,65
FILL 1,1,67
FILL 1,1,71
FILL 1,1,79
FILL 1,1,95
FILL 1,1,127
```

puis

```
CURSET 120,100,1
FILL 10,1,1
```

Il est impossible de passer en revue toutes les caractéristiques offertes par cette commande, mais voici un aperçu de quelques propriétés que vous complèterez, par expérimentation, selon vos besoins.

Les commandes PAPER ou INK annulent l'action des attributs contrôlant le papier ou l'encre.

```
HIRES
FILL 20,10,20
PAPER 4
```

Un FILL colorant le papier ($16 \leqslant X \leqslant 23$) est effacé par un ordre FILL ayant un paramètre X compris entre 0 et 127.

```
HIRES
CURSET 0,0,1
FILL 10,1,22
CURSET 0,0,1
FILL 10,3,63
```

Pour X compris entre 32 et 127, vous obtenez des blocs de pixels sur lesquels vous pouvez effectuer des changements d'encre :

```
HIRES
FILL 10,5,100
INK 4 : PAPER 1
```

Vous constatez que le premier bloc ne change pas de couleur, que le second disparaît et que c'est seulement à partir du 3ème bloc que la couleur de l'encre change.

De plus le changement de couleur de papier supprime le premier bloc. Maintenant si vous tapez :

```
HIRES
CURSET 0,0,1
FILL 10,5,63
CURSET 0,0,1
FILL 10,1,22
```

Vous voyez que seul le premier bloc (issu de la première commande FILL) se colore sous l'action d'un ordre FILL comportant un attribut de couleur de papier. Les autres blocs ne changent pas.

Voici un programme, montrant à la fois l'action des divers paramètres X, qui concluera ce paragraphe sur la commande FILL.

```
10 HIRES
20 FOR A = 1 TO 128
30 IF A>23 AND A<32 THEN60
40 CURSET RND(1)*100+1,RDN(1)*100 +1,1
50 FILL RND(1)*60+1,1,A
60 NEXT A
```

Cela vous montre d'ailleurs, qu'il n'est pas nécessaire de fournir des paramètres entiers. CURSET, comme FILL, travailleront avec la partie entière.

A vous maintenant d'entreprendre vos propres manipulations pour maîtriser et comprendre l'action des diverses commandes du mode HIRES.

Pour terminer cette partie sur les commandes, indiquons comment on peut obtenir certains effets spéciaux en mode LORES ou TEXT.

Par exemple, on peut faire imprimer les caractères en double hauteur. Faites [CTRL] [D]. Tout ce que vous écrirez sera désormais doublé en hauteur. Pour annuler cet effet, tapez [CTRL] [D]. Vous pouvez obtenir le même effet en tapant :

```
PRINT CHR$(4)
```

Vous trouverez en annexe la liste des caractères de contrôle et de leur action (nous venons de voir celle du caractère de contrôle D) Vous pouvez également faire clignoter des caractères en utilisant le petit programme suivant :

```
 5 INK 1
10 PRINT CHR$(12)
20 PRINT CHR$(27);"N BONJOUR"
30 PRINT CHR$(27);"N BONJOUR"
```

La ligne 20 comporte l'instruction produisant le clignotement. On retrouve l'expression CHR$(27) rencontrée dans un paragraphe précédent. Cette fois elle est couplée à la lettre N, comme l'indique la ligne 20.

La ligne 10 est équivalente à CLS ; elle est indispensable pour la bonne exécution du programme. Vérifiez-le en la supprimant.

Nous avons déjà vu que par la commande PLOT on pouvait également obtenir un clignotement.

Tapez :

```
 1 CLS
 2 TEXT
 5 FOR N=1 TO 24
10 PRINT "TEXTE CLIGNOTANT"
20 NEXT N
30 PLOT 5,12,12
```

Voici un programme qui va faire intervenir quelques-unes des commandes étudiées précédemment et qui va clore en le résumant par un dessin, cette partie du chapitre consacrée aux commandes.

```
 10 HIRES
 20 X=40
 30 CURSET 80,100,3
 40 PAPER 6 : INK 1
 50 STP =2*PI/40
 60 F = 0
 70 REPEAT
 80 DRAW X*COS(F),-X*SIN(F),1
 90 DRAW -X*SIN(F),-X*COS(F),1
100 DRAW -X*COS(F),X*SIN(F),1
110 DRAW X*SIN(F),X*COS(F),1
120 F=F+STP
130 X = X-1
140 UNTIL F >2*PI
150 CURSET 120,80,1
160 REPEAT
170 READ A,B
180 DRAW A,B,1
190 UNTIL B=0
200 CURSET 170,40,3
210 DRAW 10,-10,1
220 CURSET 160,20,3
230 FOR N=1 TO 10
240 READ L
250 CHAR L,0,1
260 CURMOV 7,0,3
270 NEXT N
500 DATA 50,10,-20,-20,-100,-20,60
510 DATA -40,20,10,70,10,-20,0
520 DATA 69,78,32,65,86,65,78,84,32,33
```

Si vous vous reportez à l'annexe G décrivant la carte mémoire d'Oric, vous vous apercevrez que les caractères standard ou alternés sont stockés dans la mémoire vive (ou RAM) d'Oric. On peut donc accéder à l'endroit de la mémoire où sont rangés ces caractères.

En fait toutes les informations servant à définir un caractère sont rangées par octet (groupe de 8 bits 0 ou 1). Il faut 8 octets pour décrire le caractère. A chacun de ces octets correspond un nombre, qui est l'adresse de l'octet et qui permet de le repérer dans la mémoire.

Un caractère occupera donc 8 adresses dans la mémoire. L'ensemble des adresses est compris entre les nombres 46080 et 47104, soit 1024 adresses, ce qui correspond bien à 128 caractères 128 x 8 = 1024.

Les 8 octets définissant un même caractère ont des adresses successives. Ainsi A est contenu dans les octets ayant les adresses :

46600	0 0 0 0 1 0 0 0)	
46601	0 0 0 1 0 1 0 0)	
46602	0 0 1 0 0 0 1 0)	
46603	0 0 1 0 0 0 1 0)	8 octets définissant A
46604	0 0 1 1 1 1 1 0)	
46605	0 0 1 0 0 0 1 0)	
46606	0 0 1 0 0 0 1 0)	
46607	0 0 0 0 0 0 0 0)	

adresses

Vous voyez que la disposition des 1 parmi les 0 forme l'image d'un A. A l'impression, Oric fera correspondre les 1 à de l'encre et les 0 à du blanc.

Les caractères sont rangés dans l'ordre de leur code ASCII. Ainsi le caractère @ dont le code ASCII est 64 est stocké entre les adresses 46080 + (64 * 8) et 46599 = 46080 + (64 * 8) + 7. A, dont le code ASCII est 65, est stocké aux adresses suivant 46599, soit de 46600 à 46607 (ou de 46080 + (65 * 8) à 46080 + (65 * 8) +7).

Nous allons maintenant prendre connaissance de deux commandes, qui permettent pour la première, de voir ce qu'il y a en mémoire à telle ou telle adresse, et pour la seconde de modifier ce qu'il y a en mémoire.

A l'adresse 46600 est rangé le premier octet définissant le caractère A ; il vaut en binaire 00001000, soit 8 en décimal. C'est pourquoi, si vous tapez PRINT PEEK (46600), vous obtiendrez : 8.

PEEK, suivi du nombre adresse de l'octet, vous fournit donc le contenu de cet octet (en décimal), et peut donc vous permettre de savoir quels sont les octets stockés à n'importe quelle adresse.

POKE vous permet de modifier l'octet rangé à une certaine adresse K.

Si vous tapez POKE 46600,15 vous vous apercevrez que vous avez modifié la forme géométrique du caractère A, en modifiant la valeur d'un octet définissant ce caractère.

46600 : 00001000 est devenu
46600 : 00001111

et la variation géométrique de A correspond à la variation de la disposition des 1 dans l'octet 46600.

Vous avez donc maintenant la possibilité de transformer tous les caractères à votre guise. Si vous avez de la patience et du talent vous pourrez même obtenir un alphabet gothique ou russe. Et si la fantaisie vous en prend, vous pouvez même supprimer tous les caractères en tapant POKE K,0 pour K allant de 46080 à 47104.

Précisons, cependant, qu'après POKE la valeur décimale de l'octet que vous introduisez doit être comprise entre 0 et 63, ce qui correspond en binaire aux nombres de 6 bits au plus (de 0 à 111111).

En effet, pour tous les caractères, seuls les six derniers bits de chaque octet définissant le caractère, contiennent les informations géométriques servant à dessiner ce caractère.

Voici maintenant un programme qui va vous être très utile pour redéfinir le caractère de votre choix comme vous le désirez.

```
   1 REM *** MODIFICATIONS DE CARACTERES ***
  10 CLS
  20 DIM X(8):DIM Y(8,8):DIM L(8)
  30 PRINT "ENTREZ LE CARACTERE QUE",
     "VOUS SOUHAITEZ MODIFIER"
  40 GET A$
  50 PRINT A$
  60 C = ASC (A$)
  70 A = 46080 : D = C * 8
  90 GOSUB 1000
 100 PRINT "ENTREZ HUIT NOMBRES"
 110 PRINT "(UN PAR LIGNE)"
 120 PRINT "COMPOSES CHACUN DE"
 122 PRINT "SIX ZEROS OU UNS"
 130 FOR N = 0 TO 7
 150 INPUT X(N)
 160 FOR P = 6 TO 1 STEP -1
 170 LA = INT(X(N)/(10↑(P-1)))
 180 X(N) = X(N)-(10↑(P-1)) * LA
 190 L(N) = L(N)+LA*(2↑(P-1))
 200 NEXT P
 210 X(N) = L(N)
 220 POKE(A+D+N),X(N)
 230 NEXT N
 240 GOSUB 1000
 250 STOP
1000 REM *** SUB.MODIF.CARAC. ***
1010 FOR N = 0 TO 7
1020 X(N) = PEEK (A+D+N)
1030 FOR M = 0 TO 7
1040 Y(N,M) = INT(X(N)/2↑(7-M))
1050 Z = ((X(N)↑2 (7-M))-Y(N,M))*2↓(7-M)
1060 X(N) = Z+0.0001
1080 IFY(N,M)= 0 THEN POKE 48220+(N*40)+M,32
1090 IFY(N,M)= 1 THEN POKE 48220+(N*40)+M,128
1100 NEXT M
1110 NEXT N
1120 RETURN
```

Commentaires :

Le tableau à deux dimensions : Y(8,8) va stocker un caractère (sous forme de 8 octets).

Le tableau à une dimension X(8) va décrire successivement chaque octet.

On demande tout d'abord de rentrer un caractère (celui que vous souhaitez modifier). On cherche alors l'adresse mémoire de ce caractère grâce à son code ASCII.

Vous introduisez ensuite 8 nombres binaires représentant (écrits les uns au dessus des autres) par la disposition des 1, la nouvelle géométrie du caractère.

Ces nombres binaires seront alors traduits en décimal, avant d'aller modifier (par POKE) les anciens octets.

Les lignes 1080 et 1090 font afficher le nouveau caractère (à grande échelle) sur l'écran.

32 (caractère blanc) correspondant ici à 0
et 128 (bloc plein) correspondant ici à 1

Mais les adresses mémoire ne concernent pas uniquement les caractères. Elles concernent aussi les adresses liées à l'écran ; chaque région de l'écran correspond à une adresse mémoire.

Si vous vous reportez à la carte mémoire d'Oric, vous trouverez les adresses mémoires concernant les lieux géométriques de l'écran, en mode TEXT et en mode HIRES. Il y en a beaucoup plus en mode HIRES, parce que la résolution est plus fine dans ce mode et que l'on a besoin de davantage d'adresses pour se repérer.

Nous allons utiliser ces adresses tout-à-l'heure.

Auparavant, il est nécessaire de reparler des attributs que nous avons déjà rencontrés pour les commandes PLOT (en mode TEXT ou LORES) et FILL (en mode HIRES).

Ces attributs, dont le rôle est défini en annexe, sont toujours compris entre 0 et 31 et ont les mêmes actions, utilisés avec la commande POKE, ou avec les commandes PLOT et FILL.

Tapez par exemple :

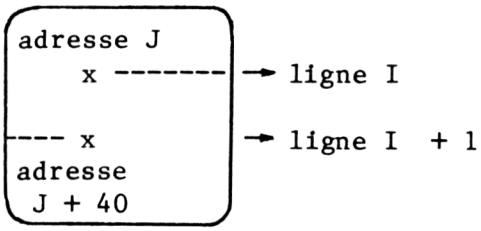
```
10 FOR N + 1 TO 24
20 PRINT "TEXTE POUR REMPLIR L'ECRAN"
30 NEXT N
50 POKE 48162,12
```

Vous voyez alors une ligne clignoter. 48162 est une adresse mémoire correspondant au lieu de l'écran où le clignotement apparaît, et 12 est l'attribut qui provoque le clignotement.

Dans le programme ci-dessus, vous pouvez changer cet attribut et vérifier que son action, lorsqu'il varie de 0 à 31, concorde avec celle décrite par les annexes.

Si dans la ligne 50 vous changez 48162 en 48200, vous constaterez que l'action de l'attribut se décale d'une ligne vers le bas.

En effet, en mode TEXT ou LORES, il y a 40 cellules par ligne et à chacune d'elle correspond une adresse. Pour changer de ligne il faut donc ajouter 40 à l'adresse mémoire.



De même, en mode HIRES, nous avons vu avec FILL que les pixels (les points) n'étaient pas pris en compte individuellement, mais qu'Oric les considérait groupés par bloc de six.

Comme il y a 240 pixels par ligne, il y a 40 de ces blocs, et à chacun de ces blocs correspondra une adresse mémoire.

Il y a donc, en mode HIRES, comme en mode LORES ou TEXT, 40 adresses mémoire par ligne.

Tapez :

```
HIRES
POKE 41000,21
POKE 41040,22
```

Mais les attributs ne constituent pas les seuls paramètres pouvant être placés après l'ordre POKE.

En fait, un octet, de par la façon dont les 0 ou les 1 qui le composent sont disposés, conditionne la manière de faire imprimer un caractère ou afficher une couleur sur l'écran.

Suivant la forme binaire de ces octets, on a affaire à un attribut ou à un modèle (PATTERN) qui peut servir, comme on l'a vu, à construire des traits interrompus.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Si les bits 5 et 6 sont tous les deux égaux à 0, alors l'octet sera lu comme un attribut.

La valeur de cet attribut est donnée par les cinq derniers bits (4 à 0), c'est-à-dire qu'en binaire, les attributs sont compris entre 00000 et 11111, soit en décimal entre 0 et 31.

Si par contre l'un des deux bits (5 ou 6) n'est pas nul, l'octet sera lu comme un modèle : le nombre modèle ayant pour valeur les six derniers bits (5 à 0) soit en décimal les valeurs 32 à 63.

Ce sont ces nombres modèle que nous avons utilisés pour construire des traits interrompus avec DRAW, ou dessiner des blocs grâce à FILL. Nous avions utilisé des modèles supérieurs à 63, mais en fait ces nombres supérieurs à 63 ont plus de 6 chiffres en représentation binaire.

Pour FILL, par exemple, seuls comptaient les six premiers 0 ou 1 du nombre (en binaire). Et avec six 0 ou 1, on ne peut dépasser 63.

Les sept premiers bits permettent de construire des nombres binaires allant jusqu'à 127, soit 01111111 en binaire.

Le bit 7 est resté égal à 0 jusque là. Si on le prend

égal à 1, on passe alors à des nombres supérieurs à 127, allant jusqu'à 255. Le bit 7 égal à 1 va produire une inversion des caractères, comme vous allez le voir maintenant.

Les petits programmes suivants vont nous montrer quelle est l'action des divers octets après l'ordre POKE, qu'ils soient considérés comme attributs ou non.

```
10 FOR N = 1 TO 24
20 PRINT "TEXTE POUR REMPLIR L'ECRAN"
30 NEXT N
40 FOR X = 0 TO 31
50 POKE 48202,X
55 WAIT 25
60 NEXT X
```

Attention, les attributs 24 à 31 dérèglent la synchronisation de l'écran.

Passez maintenant aux octets suivants, en remplaçant la ligne 40 par :

```
FOR X = 32 TO 127
```

puis faites exécuter à nouveau le programme.

Enfin remplaçons, pour terminer la démonstration, la ligne 40 par :

```
FOR X = 128 TO 255
```

Par POKE, vous pouvez affecter un attribut à un endroit particulier de l'écran.

Il exerce alors son action sur toutes les cellules situées à sa droite. Et une cellule est affectée par l'attribut situé à sa gauche : c'est le premier attribut rencontré à sa gauche qui prédomine.

Que ce soit en mode TEXT ou HIRES, vous devez donc placer l'attribut à la gauche de la cellule dont vous voulez modifier la couleur de l'encre, du papier, ou dont vous voulez faire clignoter le caractère qui s'y trouve.

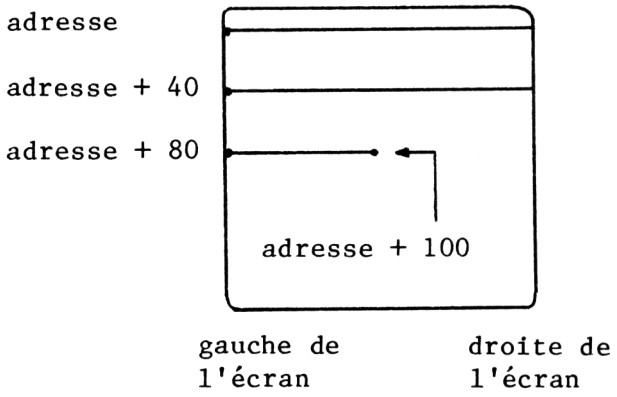
Tapez :

```
HIRES
POKE 48039,1
POKE 48040,64
POKE 48039,12
```

Le programme suivant vous montre une utilisation de POKE pour séparer l'écran en deux : une zone choisie par avance peut avoir une couleur d'encre fixe et bien déterminée, alors que l'autre zone peut avoir une couleur d'encre variable.

On choisit des attributs de couleur d'encre au hasard qu'on place par POKE aux adresses 40960 + K*40, K allant de 0 à 200. Ces adresses correspondent à l'extrême gauche de toutes les lignes disponibles en haute résolution.

Pour chacune de ces adresses, on place un attribut de couleur d'encre bien déterminé à l'adresse en question augmentée de 100, c'est-à-dire à la moitié de l'écran, deux lignes plus bas.




```
 1 HIRES
10 FOR X = 40960 TO 49079 STEP 40
30 POKE X, INT (RND(2)*7+1)
40 POKE X + 100,2
50 NEXT X
60 CURSET 120,100,0
70 FOR N = 96 TO 1 STEP -5
80 CIRCLE N,1
90 NEXT N
```

CHAPITRE **5**

Les sons et l'oric

5.1 INTRODUCTION

Votre Oric contient quelques commandes sonores particulièrement sophistiquées utilisant un circuit spécial qui peut synthétiser trois sons différents, qu'ils soient purs ou mélangés à un bruit.

Vous avez déjà fait l'expérience de quelques-uns des sons disponibles sur Oric. Chaque fois que vous pressez une touche, Oric produit un bip sonore aigu. Si vous pressez sur [RETURN] ou n'importe quelle touche de contrôle, vous entendrez un bip plus grave. Si vous voulez supprimer les sons produits par les touches du clavier, il vous suffit d'appuyer en même temps sur les touches [CTRL] et [F]. Ces sons seront restitués si vous appuyez à nouveau simultanément sur [CTRL] et [F].

Oric offre des sons prédéfinis pour les jeux (guerre de l'espace...). Il permet aussi de simuler un modèle réduit de clavier de piano, etc... Etudions maintenant toutes ses possibilités sonores.

5.2 LES SONS PREDEFINIS

Quatre sons sont préenregistrés. Il s'agit de ZAP, PING, SHOOT et EXPLODE, lesquels peuvent être obtenus très facilement. Ecrivez ZAP sur le clavier, puis appuyez sur la touche [RETURN], vous entendrez une décharge sifflante qui reproduit le son d'un pistolet galactique à laser. Maintenant, entrez de la même façon PING, vous obtiendrez une sonnerie qui peut d'ailleurs être directement produite en appuyant à la fois sur [CTRL] et [G]. SHOOT, quant à elle, est une commande simulant le tir d'un pistolet terrien à poudre. Enfin EXPLODE produit... une explosion.

Ces quatre sons prédéfinis vous seront très utiles pour le bruitage des jeux d'action que vous programmerez. Ils peuvent être inclus dans des programmes comme n'importe quelle autre commande basic.

Par exemple :

```
10 FOR N = 1 TO 10
20 ZAP
30 WAIT 5
40 NEXT N
```

produira une salve de ZAP, imitant une mitrailleuse galactique.

Vous pouvez remarquer qu'à la ligne 30, on a introduit une pause grâce à l'instruction WAIT 5. Ceci permet au son de finir avant qu'un autre ZAP ne débute. La durée du silence à mettre dépend en fait de celle du son. Un temps de pause inférieur à la durée d'exécution du son n'a aucun effet et le son produit sera continu.

Mais passons aux principales commandes de son.

5.3 LES COMMANDES DE SON

Les commandes de son sont au nombre de trois. Il s'agit de SOUND, MUSIC et PLAY. Les deux premières servent à définir la nature du son (son pur ou mélangé à un bruit) et sa hauteur, tandis que la troisième contrôle l'exécution sonore (timbre, durée d'exécution, volume).

Ces commandes nécessiteront un certain temps avant de vous devenir familières, mais elles vous donneront la possibilité de composer de la musique comme sur n'importe quel instrument existant. De plus, du bruit peut être ajouté pour imiter le son des bombes, des avions... etc. Vous verrez que les possibilités de sonorisation ne sont en fait limitées que par votre imagination et ne sont pas aussi compliquées qu'elles peuvent apparaître à première vue.

Mais avant de passer au détail des commandes de son, je vous propose ce petit programme de musique aléatoire qui vous donnera une idée des sons que l'on peut obtenir sur Oric. Pour arrêter son exécution, il vous suffira d'appuyer simultanément sur les touches [CTRL] et [C].

```
 5 REM ** MUSIQUE ALEATOIRE **
10 MUSIC 1, RND(1) * 6, RND(1) * 12 + 1, 7 +
   RND(1) * 8
20 WAIT RND(1) * 20 + 5
30 GOTO 10
```

Pour comprendre l'exécution de ce programme, il vous suffit de vous reporter aux explications concernant la commande MUSIC.

Mais examinons maintenant le détail des trois commandes sonores SOUND, MUSIC et PLAY.

5.3.1 SOUND (canal, période, volume)

Cette commande vous permet d'obtenir un son dont vous fixerez la période et le volume et qui sera exécuté tel quel ou bien mélangé à un bruit suivant le canal choisi. La commande comporte trois paramètres qui doivent être numériques (sinon la commande ne pourra être exécutée). Regardons-les en détail :

- LE CANAL

Ce paramètre peut prendre 6 valeurs, de 1 à 6. Si vous choisissez une des trois premières valeurs : 1, 2 ou 3, le son qui sera produit sera un son pur, défini par les paramètres période et volume qui suivent.

Si vous choisissez pour canal la valeur 4, 5 ou 6, vous produirez à l'exécution sonore, le son pur auquel s'ajoutera un bruit, toujours le même. L'avantage de disposer de 3 canaux de sons purs ou de sons mélangés à un bruit, est de pouvoir ainsi définir simultanément plusieurs sons (trois au plus) qui pourront être synthétisés par la commande PLAY, que nous verrons ultérieurement.

- LA PERIODE

Ce paramètre mesure la période de vibration de la note et commande de fait la hauteur du son produit. Plus la période est faible, plus la note est aiguë ; plus la période est élevée, plus la note est grave.

- LE VOLUME

Ce paramètre mesure évidemment l'intensité sonore à l'exécution. Il peut prendre 15 valeurs de 1 à 15, le volume 1 désignant l'intensité la plus faible. A moins que vous n'utilisiez un amplificateur Hi-Fi externe, vous trouverez probablement que les volumes 6 et 7 sont amplement suffisants.

REMARQUE : Vous pouvez mettre le volume à la valeur zéro. A ce moment-là l'exécution sonore du son, défini par sa période et rangé dans un canal déterminé, sera assurée par la commande PLAY suivante à condition que celle-ci renvoie au canal où a été placé le son.

En pratiquant les commandes sonores d'Oric, vous vous rendrez compte qu'il est plus commode d'utiliser SOUND seulement pour définir le son et de le faire exécuter ensuite par PLAY. Il en découle que le volume dans SOUND sera souvent mis à 0.

5.3.2 MUSIC (canal, octave, note, volume)

La commande MUSIC diffère de SOUND par le fait qu'au lieu de définir un son par sa période de vibration, elle définit une note de musique par le nom qu'elle porte et l'octave où elle se trouve. La commande MUSIC vous permet

donc de créer des sons purs d'une hauteur déterminée, correspondant tout simplement aux notes de la gamme.

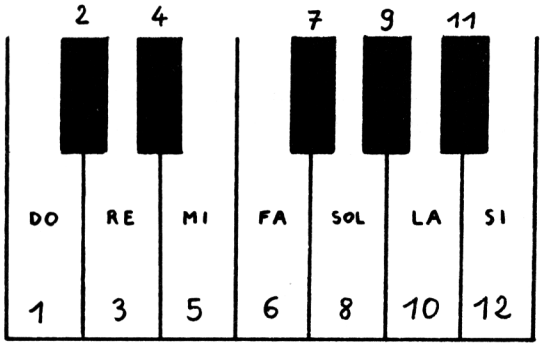
- LE PARAMETRE CANAL ne peut prendre que les valeurs 1, 2 ou 3 qui produisent des sons purs. La commande MUSIC ne peut pas mélanger une note à un bruit !

- LE PARAMETRE OCTAVE peut prendre 8 valeurs de 0 à 7, la valeur 0 donnant l'octave le plus grave. Ce nombre d'octaves permet de couvrir largement le clavier classique d'un piano.

- LE PARAMETRE NOTE peut prendre 12 valeurs de 1 à 12, et la correspondance avec les notes de la gamme est la suivante :

1 : do
2 : do # ou ré b
3 : ré
4 : ré # ou mi b
5 : mi
6 : fa
7 : fa # ou sol b
8 : sol
9 : sol # ou la b
10 : la
11 : la # ou si b
12 : si

soit encore, en reproduisant le clavier du piano :



- LE PARAMETRE VOLUME varie comme pour SOUND de 1 à 15. La mise à zéro du volume entraîne, comme pour SOUND, l'exécution de la note par une commande PLAY qui vient après.

Pour essayer cette commande MUSIC, je vous propose de monter les notes de la troisième octave. Ce programme vous permettra d'utiliser votre Oric comme un instrument à clavier puisque, enfoncées, les douze premières touches de la ligne supérieure de votre clavier produiront les notes correspondantes. Les neuf premières touches marquées de 1 à 9 produiront les notes de do à sol , de 1/2 ton en 1/2 ton, et les trois touches suivantes ("0", "-" et "=") compléteront la gamme jusqu'à si.

En appuyant sur "/", vous interromprez le programme.

```
  5 REM ** ESSAI DU CLAVIER **
 10 PLAY 0, 0, 0, 0
 20 GET A$
 30 A = VAL(A$)
 40 IF A$ = "-" THEN A = 11
 50 IF A$ = "=" THEN A = 12
 60 IF A$ = "/" THEN PLAY 0, 0, 0, 0 : STOP
 70 IF A$ = "0" THEN A = 10
 80 MUSIC 1, 3, A, 10
 90 WAIT 10
100 GOTO 10
```

Ce programme appelle quelques commentaires. La ligne 20 attend une saisie au clavier, la ligne 30 lit la valeur entrée et l'introduit dans la variable A. Si vous avez entré un chiffre (1 à 9), A aura la valeur de ce chiffre ; si vous avez entré autre chose, alors A est mis automatiquement à zéro. Les lignes 40 à 60 convertissent les touches aux valeurs requises. La ligne 70 rejette la valeur zéro pour A car MUSIC ne peut l'accepter. La ligne 90 commande la durée d'exécution du son puisqu'elle fait attendre 100 ms l'exécution du programme, durant lesquelles la note définie par MUSIC est exécutée. Au bout de ces 100 ms, on passe à l'instruction suivante PLAY 0, 0, 0, 0 qui arrête l'exécution sonore. Si elle n'y était pas, la note aurait continué à jouer jusqu'à ce que vous ayez appuyé sur une nouvelle touche.

Retenez cette instruction PLAY 0, 0, 0, 0 qui sert souvent.

Ce petit programme vous montre que l'exécution d'un son en utilisant seulement MUSIC nécessite les instructions WAIT et PLAY 0,0,0,0.

Cela aurait été la même chose avec SOUND. Pour faciliter l'exécution de sons quelconques ou de notes, on utilise donc plutôt la commande PLAY.

5.3.3 PLAY (combinaison des canaux de son pur, combinaison des canaux avec bruit, mode de sortie sonore, période)

Avant de passer au détail des paramètres de cette commande, rappelons qu'avant PLAY ont dû être définis par SOUND ou MUSIC des sons ou des notes rangés dans des canaux et dont le paramètre volume a été mis à zéro.

- LE PARAMETRE COMBINAISON DES CANAUX DE SON PUR va combiner entre eux les canaux de sons purs où se trouvent des notes ou sons purs. Il peut prendre huit valeurs de 0 à 7 dont la signification est la suivante :

0 :	aucun canal de son pur n'est branché en sortie
1 :	le canal 1 est branché en sortie
2 :	le canal 2 est branché en sortie
3 :	les canaux 1 et 2 sont branchés en sortie
4 :	le canal 3 est branché en sortie
5 :	les canaux 3 et 1 sont branchés en sortie
6 :	les canaux 3 et 2 sont branchés en sortie
7 :	les canaux 1, 2 et 3 sont branchés en sortie

- LE PARAMETRE DE COMBINAISON DES CANAUX AVEC BRUIT va combiner entre eux les canaux 4, 5 et 6 qui contiennent des sons mélangés à un bruit.

Les valeurs de ce paramètre sont similaires au paramètre de combinaison des canaux purs.

0 :	aucun canal de son mélangé n'est branché en sortie
1 :	le canal 4 est branché en sortie
2 :	le canal 5 est branché en sortie
3 :	les canaux 4 et 5 sont branchés en sortie

4 : le canal 6 est branché en sortie
5 : les canaux 6 et 4 sont branchés en sortie
6 : les canaux 6 et 5 sont branchés en sortie
7 : les canaux 4, 5 et 6 sont branchés en sortie

- LE MODE DE SORTIE

Le troisième paramètre de PLAY va définir le mode de sortie sonore. Sept modes de sorties numérotées de 1 à 7 sont disponibles. Il peuvent être schématisés comme suit.

1 :

2 :

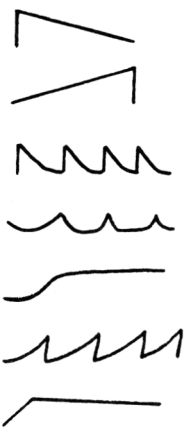
3 :

4 :

5 :

6 :

7 :



Ce paramètre permet de créer des sons légèrement montants ou descendants, continus ou oscillants. Ces caractères sont plus ou moins marqués suivant la valeur prise par le quatrième et dernier paramètre de PLAY.

- LA DUREE DU SON

Le quatrième paramètre de PLAY est un chiffre allant de 0 à 32767. Ce chiffre contrôle la durée du son de la façon suivante. Si le mode de sortie n'est pas périodique (modes de sortie 1, 2, 5, 7), il mesure simplement la durée du son et plus cette durée est grande, plus le caractère montant ou descendant du mode de sortie en question sera accusé. Par contre, si le mode de sortie est périodique (modes 3, 4, 6), le paramètre mesure alors la période du mode de sortie (à ne pas confondre avec la période propre du son, liée à sa hauteur). Dans ces cas de sorties, plus le paramètre est élevé, plus la période est étalée et plus les variations sonores s'entendent bien.

En fait ces modes de sortie vous deviendront d'autant plus familiers que vous les pratiquerez. Pour commencer votre apprentissage, je vous propose ce petit programme qui vous permet de manipuler les paramètres mode de sortie et période sur la note **la** :

```
 5 REM ** ESSAIS DE SORTIE DU LA **
10 INPUT "ENTREZ LE MODE DE SORTIE DE 1 à 7"; M
20 IF M < 10 OR M > 7 THEN 10
30 INPUT "ENTREZ LA PERIODE DE 0 A 32767"; P
40 IF P < 0 OR P > 32767 THEN 30
50 MUSIC 1, 3, 10, 0
60 PLAY 1, 0, M, P
70 PRINT "APPUYEZ SUR RETURN SI LE SON CONTINUE"
80 GOTO 10
```

Vous interromprez ce programme en appuyant sur [CTRL] et [C].

L'intérêt de la commande PLAY tient dans ces possibilités de sortie mais surtout dans le fait qu'elle peut synthétiser jusqu'à 3 sons purs et 3 sons mélangés à un bruit.

Ainsi :
```
10 MUSIC 1, 3, 1, 10
20 MUSIC 2, 3, 5, 10
30 MUSIC 3, 3, 8, 10
40 PLAY 7, 0, 1, 500
50 WAIT 600
60 PLAY 0, 0, 0, 0
```

produit l'accord do, mi, sol.

Pour terminer ce chapitre, voici une petite mélodie qui vous montrera une façon de rentrer les notes et leur durées grâce aux instructions DATA :

```
 10 REM ** MELODIE **
 20 FOR N = 1 TO 11
 30 READ A, B
 40 MUSIC 2, 3, A, 0
 50 PLAY 3, 0, 7, 2000
 60 WAIT B
 70 PLAY 0, 0, 0, 0
 80 NEXT N
 90 DATA 5, 30, 5, 30, 7, 30, 8, 75, 5, 75, 8, 60
100 DATA 10, 30, 7, 60, 5, 30, 3, 30, 5, 180
```

Annexes

A) Récapitulation des fonctions et commandes au Basic p 145

B) Les messages d'erreur p 171

C) Utilisation d'une cassette p 178

D) Utilisation d'une imprimante p 181

E) Numérotation hexadécimale, représentation des nombres p 183

F) Tableau des codes ASCII p 187

G) Carte mémoire d'Oric p 188

H) Rôle des attributs p 189

Annexe A : RECAPITULATIF DES FONCTIONS ET DES COMMANDES DU BASIC DE L'ORIC

Rappelons, avant de commencer, ce qui distingue une fonction d'une commande.

Une commande est constituée d'un mot-clé éventuellement suivi par un ou plusieurs paramètres séparés par des virgules, tandis qu'une fonction est constituée d'un mot-clé qui est le nom de la fonction, obligatoirement suivi par une ou plusieurs variables appelées arguments de la fonction, séparés par des virgules et placés entre parenthèses. Une commande va exécuter une chose bien précise en fonction des valeurs des paramètres données.

Par exemple tapez :

```
N = 1
PAPER N
```

Cela crée un fond rouge pour l'écran. PAPER est le nom de la commande et N son paramètre, qui vaut ici 1.

Une commande qui est employée à l'intérieur d'un programme prend le nom d'instruction du programme.

Une fonction, elle, va calculer une grandeur à partir des valeurs de ses variables au moment où elle est appelée. Une fonction peut être considérée comme une commande n'exécutant que des calculs.

Par exemple, la fonction racine carrée sera définie par SQR(X) (SQR est l'abréviation de square root : racine carrée). SQR est le mot-clé désignant la fonction racine carrée, X est l'argument de cette fonction, c'est-à-dire dans notre exemple, une variable numérique dont la valeur doit être positive, car SQR(X) est définie par SQR(X) * SQR(X) = X qui ne peut jamais être négatif car le produit d'un nombre par lui-même, i.e. son carré, est toujours positif.

L'utilité d'une fonction est de calculer des valeurs et d'assigner ces valeurs à des variables dans un programme. Prenons l'exemple suivant :

```
 5 X = 2
10 A = SQR(X)
15 ?A
```

Notez que ?A signifie PRINT A et que ce programme produit $\sqrt{2}$ = 1,414...

Le nom de la variable aurait pu être Y au lieu de X, cela n'aurait rien changé. Ce qui compte, c'est de fournir à cette variable une valeur satisfaisante. En l'occurence un nombre (et non pas une chaîne de caractères, par exemple) obligatoirement positif. Par contre, le nom de la fonction, c'est-à-dire les trois lettres de SQR, doit être scrupuleusement respecté.

Pour chacune des fonctions et commandes que nous allons maintenant récapituler, nous allons donc vous signaler leur signification pratique, la liste des arguments ou des paramètres éventuels, la nature de ceux-ci et leur domaine de définition, puis vous donner un exemple illustrant la syntaxe d'emploi.

1) ABS(X)

Cette fonction fournit la valeur absolue de l'argument X qui peut être une variable numérique quelconque comprise entre $-1,7\ 10^{38}$ et $1,7\ 10^{38}$. Si X est positif, ABS(X) est égal à X et si X est négatif, ABS(X) est égal à -X. Dans tous les cas, ABS(X) retourne toujours une valeur positive ou nulle.

Le petit programme suivant :

```
 5 X = 4
10 Y = ABS(X)
15 Z = ABS(-4)
20 W = Y-Z
25 ?W
```

fournira la valeur nulle, car Y et Z valant tous deux 4, leur différence W est nulle.

2) ASC(A$)

L'argument de cette fonction est obligatoirement une chaîne de caractères, c'est pour s'en rappeler qu'il est noté A$. Cette fonction ASC retourne la valeur du code ASCII du premier caractère de cette chaîne de caractères A$.

Rappelons que le code ASCII est un code de représentation interne à Oric de chaque caractère alphanumérique par un nombre binaire occupant la place d'un octet. La fonction ASC va donc se contenter de décoder le 1er octet de la chaîne de caractère fournie et de donner sa valeur décimale. La correspondance entre les valeurs décimales de ces octets et ce qu'ils représentent est donnée dans l'annexe F fournissant le code ASCII.

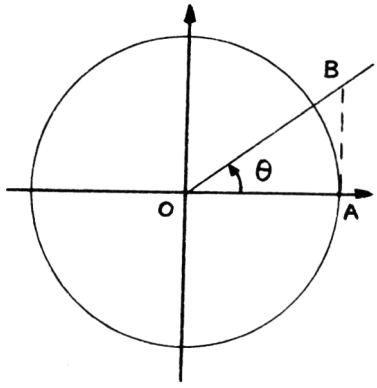
Prenons l'exemple d'utilisation suivant :

```
 5 A$ = "coucou"
10 PRINT ASC(A$)
```

ou encore en mode direct ?ASC("coucou") donnera la valeur en code ASCII de la lettre C, à savoir 67.

3) ATN(X)

Cette fonction est la fonction arctangente dont l'argument X est un nombre compris entre $-1,7\ 10^{38}$ et $1,7 * 10^{38}$. L'argument X de cette fonction représente la tangente d'un angle dont ATN(X) représente justement la valeur exprimée en radians et comprises entre $-\pi/2$ et $\pi/2$



Pour mémoire, sur le cercle trigonométrique de rayon OA unité, la longueur du segment AB représente la valeur de la tangente de l'angle θ. Si B est au-dessus de A, la tangente est positive et l'arc θ a une valeur comprise entre 0 (angle nul) et $\pi/2$ (angle droit). Si B est en-dessous de A, θ a une valeur comprise entre $-\pi/2$ et 0 et la tangente est négative.

4) CALL X

Cette commande, utilisée dans un programme, indique que la suite du programme qui se trouve à l'adresse mémoire numérotée X est écrite en code machine. On reviendra au BASIC dès que l'on trouvera l'instruction RTS.

5) CHAR X, S, F, B

Cette commande ne peut être utilisée qu'en mode HIRES. Elle consiste à dessiner un caractère à l'endroit où se trouve le curseur ; plus précisément le sommet en haut à gauche du caratère se trouvera à l'emplacement du curseur. Les trois paramètres qui suivent CHAR sont les suivants :

- X désigne le code ASCII du caractère à dessiner. C'est donc un variable numérique comprise entre 32 et 128 (cf code ASCII en annexe F).

- S est un paramètre pouvant prendre 2 valeurs : 0 ou 1.

La valeur 0 correspond au type courant d'impression tandis que la valeur 1 produira l'autre type d'impression type dit alterné, i.e. l'impression de formes géométriques permettant le dessin stylisé.

. FB est un code d'impression qui peut prendre les 4 valeurs suivantes :

0 : l'impression est faite dans la couleur du fond (numéro de PAPER) ; on ne voit donc rien mais il y a eu impression.

1 : l'impression est faite dans la couleur de l'encre (numéro de INK).

2 : inversion des couleur s.

3 : aucune impression n'est faite.

Par exemple, après être passé en mode HIRES, vous tapez :

```
CHAR 67, 0, 1
```

Cela produira la lettre C (codée par 67) en haut à gauche de l'écran. Si vous aviez tapé 1 à la place de 0, vous auriez eu ▀ qui est le caractère de l'autre type d'impression correspondant à C.

6) CHR$(X)

Cette fonction retourne le caractère dont le code ASCII est X. L'argument X de cette fonction est un nombre compris entre 32 et 128. Les fonctions ASC et CHR$ sont inverses l'une de l'autre.

Pour vous en convaincre tapez :

```
 5 INPUT "CODE ASCII 32 A 128";X
10 A$ = CHR$(X)
15 B = ASC(A$)
20 PRINT X,"=",B
```

où CHR$ et ASC pris successivement ne produisent rien.

Pour X ≤ 31, PRINT CHR$(X) exerce une action particulière qui est à rapprocher de l'action de la touche [CTRL] enfoncée simultanément avec la Xième lettre de l'alphabet. Ainsi, CHR$(4) provoque la même chose que [CTRL] [D], à savoir l'écriture sur 2 lignes à la fois.

7) CIRCLE R,FB

Cette commande ne peut être utilisée qu'en mode HIRES. Elle consiste à dessiner un cercle centré sur la position du curseur dont le rayon R est une variable numérique comprise entre 1 et 119. FB est le code d'impression vu pour CHAR.

En mode HIRES, lancez par RUN le mini-programme :

```
 5 CURSET 100,100,1
10 CIRCLE 50,1
```

Il vous dessinera une cercle au milieu de l'écran. Vous pourrez remarquer que la commande ne fonctionne pas si une partie du cercle quitte l'écran.

8) CLEAR

Cette commande met toutes les variables à zéro et vide les chaînes de caractères.

9) CLOAD "XX"

Commande de chargement d'un programme enregistré sur cassette. Une fois lu, le programme précédent est perdu.

Exemple :

```
CLOAD"TARZAN"
```

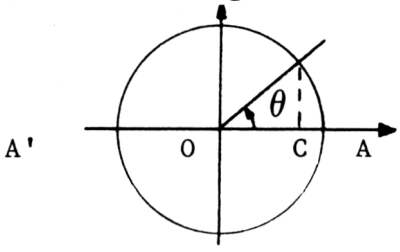
10) CONT

Cette commande continue l'exécution d'un programme après une interruption (opérée par [CTRL] et [C] ou par STOP).

Si durant l'interruption des modifications ont été apportées (notamment correction d'instruction DIM), l'exécution du programme peut ne pas reprendre par CONT. Essayez alors un GOTO numéroté, sinon relancez l'exécution au début du programme.

11) COS(X)

Cette fonction est la fonction cosinus dont l'argument X est un nombre représentant la valeur exprimée en radians de l'angle dont COS(X) est le cosinus.



Pour mémoire, sur le cercle trigonométrique unité, OC représente le cosinus de l'angle θ compris entre 1 et 0 tant que C reste entre A et O, et compris entre 0 et -1 dès que C passe entre O et A'.

Exemple :

```
 5 X = 1.5707963
10 ? COS(X)
```

donnera 0, car X vaut $\pi/2$.

12) CURMOV X,Y,FB

Cette commande place le curseur à une nouvelle position définie par ses coordonnées X et Y relatives à l'ancienne position. FB est le code d'impression vu pour CHAR.

Cette commande ne peut être utilisée qu'en mode HIRES.

13) CURSET X,Y,FB

Cette commande place le curseur à la position définie par les coordonnées absolues X et Y, avec X compris entre 0 et 239, et Y compris entre 0 et 199.

FB est le code d'impression déjà vu pour CHAR.

14) CSAVE "XX"

Cette commande préserve le fichier portant le nom XX pour l'enregistrer sur une cassette.

15) DATA

Cette instruction est suivie par une liste de constantes aussi bien numériques qu'alphanumériques et qui sont stockées dans l'ordre d'écriture et seront lues dans le même ordre par des instructions READ. Cette instruction ne peut être employée qu'en mode différé. Les constantes qui suivent doivent impérativement être séparées par des virgules.

Exemple :

```
 5 READ A,B,C,D$
10 DATA 1,2,3,"NOUS IRONS AU BOIS"
15 PRINT A,B,C,D$
```

16) DEEK (X)

Cette fonction retourne le contenu d'une octet plus 256 fois le contenu de l'octet suivant.

Exemple :

```
?DEEK(4062)
```

17) DEF FN

Cette instruction définit pour l'utilisateur une fonction numérique à une variable en faisant suivre DEF FN du nom de la fonction, de l'argument placé entre parenthèses, du signe égal suivi de l'expression analytique de la fonction.

Exemple :

```
10 DEF FNAA(X)=A*X+B
```

définit la fonction ax+b.

18) DEF USR

Cette instruction permet de définir l'adresse en mémoire du début d'un programme écrit en code machine.

Exemple :

```
DEF USR = #400
```

19) DIM

Cette instruction dimensionne tous les tableaux d'un programme, qu'ils contiennent des nombres ou des caractères alphanumériques. En pratique, cette instruction fixe les valeurs entières maximales que peuvent prendre les divers indices de tableaux. Le mot-clé DIM est suivi de la liste des noms de tableaux et de leur(s) dimension(s), chaque groupe étant obligatoirement séparé par des virgules.

Notez qu'il est interdit de dimensionner deux fois le même tableau à l'intérieur d'un programme mais un tableau non-dimensionné est implicitement dimensionné à 10 lignes.

Exemple :

```
DIM A(5,10),B(4,2),C$(8)
```

ou encore :

```
10 INPUT N
20 DIM E(N,3)
```

20) DOKE X,V

Cette instruction range en mémoire la valeur V aux adressses de mémoire X et X+1 de la façon suivante. La partie entière de V/256 va à l'adresse X+1, et le reste à l'adresse X.

21) DRAW X,Y,FB

En mode HIRES, cette commande dessine un segment de droite dont l'origine est la position du curseur et l'extrémité le point de coordonnée X,Y mesurées à partir de l'origine du segment dessiné.

FB est toujours le code de représentation vu pour CHAR.

22) END

Cette instruction termine un programme. Elle est facultative si l'instruction END se trouve à la fin du programme.

23) EXP(X)

Cette fonction calcule l'exponentielle de l'argument X, notée mathématiquement e^x. L'argument X varie entre $-1,7 \quad 10^{38}$ et $1,7 \; 10^{38}$ tandis que la fonction varie de 0 à $1,7 \quad 10^{38}$.

?EXP(0)

donnera 1

24) EXPLODE

Cette commande produit un son prédéfini ressemblant à une explosion.

25) FALSE

Cette instruction donne la valeur nulle au résultat d'un test logique.

Par exemple le test logiqueA=(B=C) vaut 1 si B=C, et vaut 0 si B est différent de C. Si l'on a écrit A=FALSE, cela signifie que le test logique sera mis forcément à sa valeur nulle.

26) FILL B,A,N

Cette commande n'est utilisable qu'en mode HIRES, et a pour fonction de traiter graphiquement une partie de l'écran :

. B désigne le nombre de lignes traitées (de 0 à 199) ;

. A désigne le nombre de blocs de six colonnes traités (de 0 à 40) ;

. N est un paramètre de traitement graphique (de 0 à 127). De 0 à 31, N est un attribut et de 32 à 127 N est un modèle (pattern).

Pour plus de détails, s'en rapporter au chapitre des commandes graphiques élaborées.

27) FN

Ce mot-clé permet de manipuler le résultat d'une fonction prédéfinie.

Exemple :

```
PRINT FNSA(3)
```

écrit la valeur de la fonction SA(3).

28) FOR...TO
STEP/NEXT

Cette instruction crée une boucle qui répète les lignes du programme comprises entre FOR et NEXT. STEP détermine la taille de l'incrément et est pris égal à 1 par omission.

Exemple :

```
10 FOR I = 1 TO 10 STEP 2
20 PRINT I
30 NEXT I
```

29) FRE(X)

Quelle que soit la valeur de l'argument, cette fonction fournit le nombre d'octets disponibles, ce qui donne une idée de la taille mémoire libre à ce moment.

Exemple :

```
?FRE(0)
```

30) GET

Cette commande ne fonctionne qu'en mode différé. Elle attend qu'une touche quelconque du clavier soit pressée et prend en compte ce qui est tapé sans qu'on ait besoin de taper sur RETURN, ce qui différencie GET de INPUT.

L'emploi classique de GET consite à l'associer à l'instruction VAL ainsi :

```
10 GET A$
20 A = VAL(A$)
```

ce qui permet de manipuler ensuite non plus des chaînes de caractères numériques mais des variavle numériques.

Notez que [CTRL] [C] est considéré comme caractère alphanumérique et ne peut pas interrompre le programme exécutant une instruction GET.

31) GOSUB N

Cette commande provoque un branchement du programme à la ligne de numéro N où débute une série d'instructions se terminant par RETURN et formant un sous-programme (SUBROUTINE).

Mais le GOSUB ne se contente pas d'être un GOTO numéroté, il conserve aussi en mémoire l'adresse de la ligne qui le suit et à laquelle le programme reviendra après RETURN.

Exemple :

```
      GOSUB 1000
        :
1000    :
        :
      RETURN
```

32) GOTO N

Cette commande opère un branchement inconditionnel à la ligne du programme portant le numéro N qui doit donc exister dans le programme.

Exemple :

```
40  GOTO 100
      :
100   :
      :
```

33) GRAB

Cette commande assigne la place en mémoire de :

9800 à B400 (48K), ou de
1800 à 3400 (16K), pour utiliser la mémoire RAM.

34) HEX$(V)

Cette fonction calcule la valeur hexadécimale de la valeur V.

Exemple :

```
PRINT HEX$(36)
```

35) HIMEM

Cette commande abaisse le niveau supérieur de la mémoire utilisable pour les programmes basic, rendant la mémoire située au-dessus utilisable pour des programmes machine.

Exemple :

```
HIMEM # 7200
```

36) HIRES

Cette commande enclenche le mode graphique haute résolution. Le fond est mis au noir, l'"encre" est blanche, et le curseur est placé en haut et à gauche de l'écran, tandis que les lignes de texte sont maintenues dans la couleur précédente.

37) IF/THEN
... ELSE

Si l'expression suivant le IF est vraie, alors toutes les instructions suivant le THEN sont exécutées. Si l'expression est fausse, alors toutes ces instructions sont ignorées et le programme exécute les instructions suivant ELSE. ELSE peut être omis.

Exemple :

```
10 IF A   10 THEN GOTO 20
15 PRINT OK
20   :
```

38) INK N

Cette commande modifie la couleur de l'encre d'écriture dans toute la surface de l'écran. N est un entier compris entre 0 et 7.

Exemple :

```
INK 3
```

39) INPUT

Cette instruction arrête l'exécution d'un programme et attend une introduction de données avant de continuer. Les données rentrées sont assignées à des variables.

Exemple :

```
        INPUT N$,A
ou :    INPUT "AGE?";B
```

40) INT(X)

Fonction retournant la partie entière de l'expression numérique entre parenthèses, c'est-à-dire le plus grand entier inférieur ou égal à la valeur de l'expression entre parenthèses.

Exemple :

```
?INT(PI)
```

donne 3.

41) KEY$

Ce mot-clé signifie qu'ORIC attend une donnée du clavier (keyboard) mais continue l'exécution, qu'une touche ait été pressée ou pas.

Exemple :

```
X$ = KEY$
```

assignera à X$ la valeur de n'importe quelle touche pressée.

42) LEFT$(A$,N)

Cette fonction d'une chaîne de caractère A$ et d'un entier N retourne N premiers caractères en partant de la gauche de la chaîne de caractères en question.

43) LEN(A$)

Cette fonction retourne la longueur de la chaîne de caractères placée entre parenthèses.

44) LET

Cette commande, qui permet d'assigner une valeur à une variable, est en fait optionnelle.

```
LET A = 4
```

équivaut à A = 4

45) LIST

Liste les lignes spécifiées ou tout le programme sur l'écran.

Exemple :

LIST 100

liste la ligne 100 du programme, mais

LIST

liste tout le programme.

46) LLIST

Cette commande équivaut à la précédente, mais la sortie, au lieu de se faire sur l'écran, se fait sur l'imprimante.

Exemple :

LLIST 50

ou : LLIST

47) LN(X)

Cette fonction retourne le logarithme népérien de X. Par définition de cette fonction, X doit être strictement positif.

Le logarithme néperien est la fontion réciproque de l'exponentielle (cf EXP(X)).

Exemple :

?LN(1)

produira 0.

48) LOG(X)

Cette fonction retourne le logarithme décimal de X, qui doit être strictement positif.

Exemple :

?LOG(100)

donnera 2.

49) LORES N

Cette commande enclenche le mode basse résolution.

Le fond de l'écran est mis au noir. Lorsque N est égal à 0, les caractères imprimés sont les caractères standard, et lorsque N vaut 1, ce sont les caractères du type alterné - représentant des figures stylisées - qui sont reproduits.

Exemple :

```
LORES 0
```

50) MID$(Z$,A,L)

Cette fonction retourne la chaîne de caractères de longieur L débutant au Aième caractère de la chaîne de caractères Z$.

Si A$ = "ORIC", MID$(A$,3,2) retournera "IC"

51) MUSIC A,B,C,D

Cette commande produit une note de musique rangée sur le canal A (1,2 ou 3), définie par le numéro B de l'octave (0 à 7), par le numéro C de la note (1 à 12), et jouée avec un volume D (0 à 15). Pour plus de détails, se reporter au chapitre traitant des commandes de son.

Exemple :

```
MUSIC 1,3,5,10
```

52) NEW

Cette commande détruit le programme courant, ainsi que toutes les variables. Cette commande est utile lorsqu'on veut tout nettoyer avant de repartir à zéro pour un autre programme.

Exemple :

```
NEW
```

53) ON ... GOSUB

Cette commande branche le programme à la Nième étiquette d'un sous-programme qui suit le mot-clé GOSUB.

Exemple :

ON N GOSUB 200,300

testera N ;

. si N = 1, le branchement se fait à la ligne 200 ;
. si N = 2, il se fait à la ligne 300.

54) ON ... GOTO

Cette commande branche le programme à la Nième étiquette spécifiée après le mot-clé GOTO.

Exemple :

ON N GOTO 1000,2000

branchera le programme à la ligne d'étiquette 2000 si N = 2.

55) PAPER N

Cette commande change la couleur du fond de l'écran suivant la valeur de l'entier N (de 0 à 7).

Exemple :

PAPER 3

56) PATTERN X

Cette commande fait appel au générateur de pointillés pour les commandes DRAW.

X est un entier compris entre 0 et 255. Sa configuration binaire avec une succession de 0 et de 1 schématise la forme du pointillé qui affectera le segment défini par

DRAW. Les 1 correspondent à des traits pleins et les 0 à des vides. Ainsi PATTERN 255 produira un trait plein car 255 en base 10 équivaut à 11111111 en base 2.

57) PEEK (X)

Cette fonction fournit le contenu de la case mémoire dont l'adresse est l'entier X.

Exemple :

```
?PEEK(58)
```

58) PI

Ce mot clé est la constante PI = 3,1415...

59) PING

Cette commande produit un son prédéfini ressemblant à une sonnerie.

60) PLAY A,B,C,D

Cette commande musicale définit la sortie sonore d'un son ou d'une note grâce aux quatre paramètres suivants :

. A, entier de 0 à 7, définit la combinaison des canaux purs.

. B, entier de 0 à 7, définit la combinaison des canaux bruités.

. C, entier de 1 à 7, définit le mode de sortie.

. D, entier de 0 à 32767, définit la période de sortie.

Pour plus de détails, se reporter au chapître des commandes sonores.

61) PLOT X,Y,A$

Cette commande inscrit le caractère A$ sur l'écran, en mode LORES ou TEXT, à l'endroit défini par les coordonnées X et Y. A la place de A$, on peut mettre un

attribut spécifique à certaines opérations graphiques évoluées, i.e. un nombre compris entre 0 et 31 (cf annexe H), ou le code ASCII d'un caractère.

62) POINT (X,Y)

Cette fonction retourne 0 si le point élémentaire d'affichage graphique est dans la couleur du fond de l'écran, et -1 si ce n'est pas le cas, c'est-à-dire si ce point est imprimé par dessus dans la couleur de l'"encre". X et Y sont les coordonnées absolues de ce point.

63) POKE N,V

Cette commande écrit la valeur V (0-255) à la case mémoire d'adresse N de l'écran. Si N renvoie à une case mémoire où est rangé un octet servant à définir un caractère, V est compris entre 0 et 63.

64) POP

Cette commande sans paramètre joue le rôle d'une instruction RETURN, à la différence qu'après une instruction POP, la prochaine instruction RETURN rencontrée opère un retour à l'instruction suivant non pas le dernier GOSUB, mais l'avant-dernier.

65) POS

Ce mot-clé retourne la position horizontale du curseur.

Exemple :

```
10  A = POS
```

66) PRINT

Cette commande imprime des variables, des nombres et des chaînes de caractères sur l'écran.

Notez que ? peut être utilisé à la place du mot PRINT mais sera converti en PRINT en listant le programme par LIST.

Exemple :

```
            PRINT A,B,C$
ou encore : PRINT "LE RESULTAT EST";B
```

67) PULL

Cette commande tire une étiquette de l'ensemble des boucles REPEAT.

Exemple :

```
PULL
```

68) READ

Cette commande lit une ou plusieurs données contenues dans une liste DATA et les assigne dans l'ordre de lecture aux variables qui suivent le mot-clé READ .

Exemple :

```
10  READ A,B
20  DATA 100,50
```

69) RELEASE

Cette commande assigne une aire de mémoire décrite dans la commande GRAB aux manipulations concernant l'écran en mode HIRES.

70) REM

Ce mot-clé permet d'introduire des commentaires,entre des instructions,dans un programme. Tout ce qui suit le mot REM est ignoré par l'interpréteur.

Exemple :

```
10  REM IGNORE CELA
```

71) REPEAT ... UNTIL

Cette commande crée une boucle pour répéter toutes les lignes du programme jusqu'à une instruction où il est procédé à un test. Si celui-ci est faux, l'exécution refait un tour et si le test est vrai, l'exécution est poursuivie à la ligne suivant UNTIL.

72) RESTORE

Cette commande replace le pointeur de lecture d'une instruction READ à la première donnée de la liste DATA.

Exemple :

```
 5 READ A,B
10 DATA 10,20
15 RESTORE
20 READ C
25 ?C
```

fournira à C la valeur 10.

73) RETURN

Cette commande indique la fin d'un sous-programme et renvoie la suite du déroulement de l'exécution à l'instruction suivant la dernière instruction GOSUB lue.

74) RIGHT$ (A$,N)

Cette fonction retourne les N derniers caractères en partant de la droite, d'une chaîne de caractère A$.

75) RND(X)

Cette fonction fournit automatiquement un nombre aléatoire chaque fois qu'elle est appelée. Si $X \geqslant 1$, alors RND(X) se trouve entre 0 et 1. Si $X = 0$ le nombre généré en dernier est produit, si $X < 0$, le nombre produit est le même pour chaque X.

76) RUN

Cette commande lance l'exécution d'un programme en basic.

77) SCRN (X,Y)

Cette fonction retourne le code ASCII pour un caractère placé à la position de coordonnées (X,Y) en mode LORES ou TEXT.

78) SGN(X)

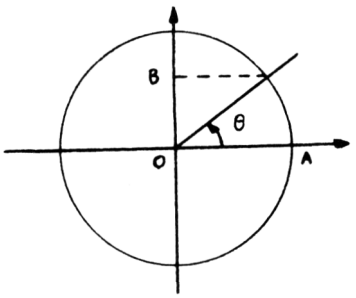
Cette fonction retourne -1 si l'argument X est négatif et 0 si l'argument X est positif ou nul.

79) SHOOT

Cette commande produit un son prédéfini imitant le bruit d'un coup de feu.

80) SIN X

Cette fonction calcule le sinus de l'angle dont la valeur en radians est X.



Pour mémoire, sur le cercle trigonométrique unité, la longueur du segment OB représente la valeur du sinus. Si B est au-dessus de A, le sinus est positif et l'angle varie entre 0 et π.
Si B est en-dessous de O, le sinus est négatif et l'angle varie entre 0 et $-\pi$.

81) SOUND A,B,C

Cette commande musicale produit un son défini par :

. A : numéro du canal où est rangé le son (1 à 6) ;

. B : période de vibration du son ;

. C : volume sonore (0 à 15).

82) SPC (N)

Cette fonction laisse un espace de N positions sur l'écran où N est un entier de 0 à 255. On l'utilise en impression, par exemple :

```
PRINT "AT"SPC(3)"CHOUM"
```

produira : AT CHOUM

83) SQR(X)

Cette fonction fournit la racine carrée de l'argument X, c'est-à-dire le nombre positif dont le produit par lui-même donne X, lequel doit obligatoirement être positif ou nul.

Exemple : ?SQR(3)

fournit : 1.732...

84) STOP

Cette instruction arrête l'exécution d'un programme et provoque un message BREAK IN 10 si, à la ligne 10, il y a 10 STOP.

85) STR$(N)

Cette fonction convertit un nombre en la chaîne de caractères correspondante. Par exemple STR$(10) convertit la constante numérique 10 en la chaîne de deux carctères 1 et 0 accolés, soit 10. Cette fonction est utile pour écrire des nombres au milieu de l'écran, car on utilise PLOT qui ne peut écrire que des chaînes de caractères, d'où la nécessité de transformer ces nombres en chaînes de caractères, ce qui est assuré par STR$.

86) TAB(N)

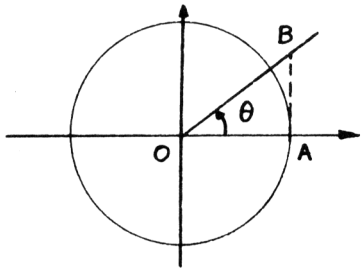
Cette commande tabule l'impression, c'est-à-dire la déplace de N positions à partir de la gauche de l'écran. N ne doit pas dépasser 249.

Exemple :

```
PRINT TAB(3)
```

87) TAN(X)

Cette fonction calcule la valeur de la tangente de l'angle dont la valeur en radians vaut X.



Pour mémoire, rappelons que dans le cercle unité, la longueur AB mesure la tangente de l'angle θ . On ne peut pas avoir X = $\pi/2$ = 1,5707... car la tangente serait infinie et n'est donc pas définie.

88) TEXT

Cette commande enclenche le mode TEXT.

89) TROFF

Cette commande supprime le suivi (trace) de l'exécution qui consiste à faire imprimer le numéro de la ligne où passe l'interpréteur au moment de l'exécution.

90) TRON

Cette commande enclenche le suivi (trace de l'exécution du programme en affichant le numéro de la ligne de chaque instruction exécutée sur l'écran.

91) TRUE

Ce mot-clé attribue la valeur -1 au résultat d'un test logique (réciproque de FALSE).

Exemple :

```
A = TRUE
```

92) USR(N)

Cette commande appelle un sous-programme en langage machine en lui transmettant l'argument N.

93) VAL(N$)

Cette fonction transforme une chaîne de caractères numériques en la valeur correspondante.

Exemple :

```
A = VAL(A$)
```

Si A$ n'est pas une chaîne de caractères numériques, VAL(A$) vaut 0.

94) WAIT N

Cette commande provoque une pause, dans l'exécution du programme, d'une durée de N centisecondes. Il est impossible de provoquer une interruption de l'exécution d'une pause par [CTRL] [C].

Exemple :

```
WAIT 100
```

signifie faire une pause d'1 seconde.

95) ZAP

Cette commande produit un son prédéfini imitant celui d'un "pistolet à laser".

Annexe B LES MESSAGES D'ERREUR

Votre Oric contient un interpréteur qui vérifie systématiquement chaque instruction basic juste avant de l'exécuter. S'il n'arrive pas à comprendre quelque chose dans cette instruction ou s'il y détecte une erreur précise, il va délivrer aussitôt ce qu'on appelle un message d'erreur. Ces messages d'erreur sont en nombre fini et Oric en contient vingt. Mais avant de les examiner un par un, il faut noter la légère différence qui existe entre les messages d'erreur délivrés lorsque l'on travaille en mode direct et ceux délivrés lorsque l'on travaille en mode différé (i.e. lorsque l'on écrit un programme). Dans les deux cas, les messages d'erreur sont les mêmes ; seulement, si vous êtes en mode direct au moment où l'erreur s'est produite, l'interpréteur se contente de délivrer le seul message d'erreur.

Par exemple, l'enregistrement :

```
PRIN "BONJOUR"
```

provoquera immédiatement le simple message d'erreur suivant : SYNTAX ERROR, c'est-à-dire ERREUR DE SYNTAXE.
Comme vous l'avez remarqué, il a été oublié le T de PRINT.

Par contre, si vous êtes en mode différé au moment où l'interpréteur détecte une erreur, celui-ci vous délivre non seulement le message d'erreur approprié, mais il inscrit de plus, à sa suite, le numéro de la ligne où il a détecté l'erreur.

Par exemple, le mini-programme :

```
5 PRIN "BONJOUR"
```

lancé par RUN, provoquera le message d'erreur suivant :

```
SYNTAX ERROR IN 5
```

Ce renseignement supplémentaire peut faire gagner beaucoup de temps, mais il faut, ici, faire la remarque suivante. Ce n'est pas parce que l'interpréteur a détecté une erreur à une certaine ligne qu'une correction devra être apportée, forcément, à cette ligne. L'erreur peut provenir d'instructions précédentes qui, bien que correctes du point de vue basic, calculent des choses fausses.

Par exemple, le programme :

```
 5 A = INT(0.5)
10 B = 10
20 C = B / A
30 PRINT C
```

provoquera inévitablement le message d'erreur suivant

```
DIVISION BY ZERO ERROR IN 20
```

La correction peut évidemment être apportée à la ligne 20, si au lieu de diviser B par A, on voulait faire le produit B * A ; mais l'erreur peut aussi provenir de la variable A, qui ne devrait pas avoir la valeur nulle si au lieu de A = INT(0.5), on avait écrit A = INT(0.5) + 1. Retenez de ce petit exemple que l'erreur n'est pas forcément là où le message d'erreur la situe. C'est à vous de voir dans chaque cas.

Au moment où vous débutez votre apprentissage informatique sur Oric, vous avez l'occasion de rencontrer plus d'une fois chacun de ces messages d'erreur.

Mais c'est en sachant d'autant mieux ce qu'ils signifient que vous apprendrez à corriger d'autant plus vite vos erreurs, et que vous les rencontrerez de moins en moins souvent. Lorsque l'interpréteur ne publie plus aucun message d'erreur, cela signifie seulement que ce que vous avez écrit est cohérent du point de vue basic, et est exécutable par Oric ; cela ne signifie pas pour autant qu'il exécute ce que vous vouliez lui faire exécuter.

Voici les vingt messages d'erreur possibles, qui sont délivrés en anglais par Oric. Ils sont classés par ordre alphabétique. Nous vous donnons ici leur traduction, leur signification et quelques indications sur les origines possibles des erreurs détectées.

1) BAD SUBSCRIPT (mauvais indice)

On a cherché un élément de tableau qui ne peut pas exister, vu la dimension qui a été donnée pour ce tableau. Par exemple A a été dimensionné par DIM A(20,20) et l'on cherche A(24,25), ce qui est en contradiction avec la dimension fournie, ou encore, ce qui est pire, on manipule A(24,25) alors que A avait été dimensionné par DIM A(5).

Dans ce cas, il vous faut revoir la compatibilité entre les valeurs des indices des éléments de tableau présents dans la ligne du programme qui a provoqué le message d'erreur, et les instructions dimensionnant ce ou ces tableaux.

2) BAD UNTIL (mauvais UNTIL)

L'interpréteur vient de lire une instruction UNTIL sans avoir lu auparavant une instruction REPEAT. Il vous reste à ajouter le REPEAT manquant ou à supprimer l'UNTIL de trop.

3) CAN'T CONTINUE (on ne peut continuer)

Oric essaie en vain de continuer ou de reprendre l'exécution d'un programme. Ce message peut paraître dans deux cas. Lorsqu'Oric a détecté une erreur et qu'au lieu de chercher à la corriger, vous essayez de reprendre l'exécution du programme en tapant CONT.

Ce message vient aussi lorsque vous avez interrompu l'exécution d'un programme et que pendant cette interruption vous avez modifié, ajouté ou détruit une ligne du programme qui empêche la poursuite normale de l'exécution du programme. Dans ce cas, on peut parfois reprendre l'exécution par un GOTO numéroté. Remarquez que si vous avez modifié le programme, vous ne pouvez plus reprendre son exécution par CONT ; il faut utiliser un GOTO numéroté.

4) DISTYPE MISMATCH (incompatibilité entre modes d'écriture)

Oric cherche à faire des choses spécifiques à un mode d'écriture (HIRES ou TEXT) alors que l'on ne se trouve justement pas dans ce mode. Le cas classique où se produit ce message est celui où l'on cherche à dessiner dans le mode TEXT.

5) DIVISION BY ZERO (division par zéro)

Ce message peut provenir d'une variable qui n'a pas été initialisée et a la valeur nulle au moment du calcul, car toute variable non initialisée explicitement a la valeur nulle dès le début du programme.

6) FORMULA TOO COMPLEX (expression trop complexe)

L'expression contenue dans la ligne qui a provoqué ce message est trop compliquée pour Oric. Par exemple, ce message est délivré si la ligne contient plus de deux IF/THEN.

7) ILLEGAL DIRECT (interdit d'emploi en mode direct)

On a cherché à utiliser une instruction ou une commande en mode direct, à partir du clavier, alors que c'était impossible.

Par exemple et pour mémoire, les instructions INPUT, GET, DEF FN, DATA sont interdites d'emploi en mode direct.

8) ILLEGAL QUANTITY (valeur erronée)

On a cherché à calculer la valeur d'une fonction en un point qui sortait de son domaine de définition.

Par exemple :

. argument de LOG négatif ou nul
. argument de SQR négatif
. 0 élevé à une puissance négative
. indice d'un tableau négatif ou supérieur à 32767

9) NEXT WITHOUT FOR (NEXT sans FOR correspondant)

L'apparition de ce message peut provenir de plusieurs causes : mauvaise imbrication de boucles entre elles, oubli de supprimer un NEXT après une correction qui a éliminé un FOR...

10) OUT OF DATA (données épuisées)

On cherche à lire encore des données alors que celles contenues dans tous les DATA ont déjà été lues. Ce message provient généralement d'un manque de données dans les DATA par rapport à ce que demandent les instructions de lecture (INPUT ou READ).

11) OUT OF MEMORY (mémoire épuisée)

Le programme est trop long ou bien demande trop de variables, ou encore contient plus de seize boucles et GOSUB emboîtés les uns dans les autres.

12) OVERFLOW (dépassement de capacité)

Un nombre, plus grand en valeur absolue que $1,7041 \times 10^{38}$ a été élaboré, à un moment donné, dans un calcul. Pour les nombres dont la valeur absolue est inférieure à $2,9 \times 10^{-39}$ ce message d'erreur n'apparaît pas, car Oric les assimile à zéro.

13) REDIM'D ARRAY (tableau redimensionné)

Ce message d'erreur se produit dans le cas suivant : Un tableau a été déjà dimensionné et on essaie abusivement de le redimensionner, en relisant cette instruction ou en lisant une nouvelle instruction le dimensionnant.

14) REDO FROM START (recommencez au début)

Au cours d'une instruction INPUT, on a essayé de rentrer une chaîne de caractères alors qu'on demandait un

nombre. Il faut recommencer à rentrer depuis le début les valeurs demandées par l'INPUT en question.

15) RETURN WITHOUT GOSUB (retour sans GOSUB)

L'interpréteur vient de lire un RETURN alors qu'il n'avait pas lu de GOSUB le lui annonçant. Cette erreur peut provenir d'une exécution lancée par un GOTO numéroté qui a sauté le GOSUB, ou bien de l'oubli de l'instruction END en fin du programme principal suivi d'un sous-programme.

16) STRING TOO LONG (chaîne de caractères trop longue)

Ce message d'erreur se produit lorsqu'on essaie de fabriquer, par concaténation, une chaîne de caractères de plus de 255 caractères, ce qui est interdit.

17) SYNTAX ERROR (erreur de syntaxe)

Ce message sera un des plus fréquemment rencontrés par le novice en informatique. Il signifie que l'instruction qui a provoqué ce message est incompréhensible pour l'interpréteur basic. Les causes peuvent être variées : oubli de parenthèse, caractères illégaux, mauvaise ponctuation, faute d'orthographe dans un mot-clé, virgule manquante, variable d'un certain type employée à tort, etc... Il vous reste donc à relire l'instruction incriminée et à vérifier l'exactitude, du point de vue basic, de chacun des caractères écrits.

18) TYPE MISMATCH (incompatibilité entre variables numériques et alphanumériques)

Ce message se produit lorsqu'on essaie d'assigner une chaîne de caractères à une variable numérique ou vice versa, ou encore lorsque l'on fournit une chaîne de caractères comme argument d'une fonction qui n'accepte que des nombres ou vice versa.

19) UNDEF'D STATEMENT (instruction non définie)

On essaie d'accéder, en utilisant une instruction GOTO, GOSUB ou THEN, à une ligne dont le numéro est inexistant. Ce message provient souvent de corrections mal faites dans le programme. Par exemple, on a modifié

un numéro de ligne et l'on a oublié de reporter cette correction sur l'instruction GOTO qui renvoyait à cette ligne.

20) UNDEF'D FUNCTION (fonction non définie)

On fait appel à une fonction qui n'a pas été définie par l'instruction DEF FN. Cela peut provenir d'un oubli de définition de la fonction, ou peut-être tout simplement d'une mauvaise orthographe de celle-ci.

Annexe C UTILISATION DE CASSETTES AVEC ORIC

Oric ne peut garder en mémoire qu'un seul programme basic. Aussi, lorsque vous aurez écrit des programmes relativement longs, vous devrez les stocker sur cassette pour les réutiliser quand vous voudrez, sans avoir à les retaper, sur un Oric.

- CONNEXION D'UN MAGNETOPHONE A CASSETTES A UN ORIC

La face arrière du boîtier d'Oric comporte une prise DIN à sept trous pour la connexion du magnétophone. Deux solutions sont possibles :

. Si la prise de votre magnétophone comporte aussi sept trous, aucun problème ne se pose et le moteur de votre magnétophone sera commandé automatiquement par Oric, sans qu'aucune intervention manuelle ne soit nécessaire.

. Si la prise de votre magnétophone ne comporte que trois à cinq trous, seuls trois d'entre eux sont à utiliser et vous devrez donc demander à votre distributeur un câble à trois fils. Vous devrez, à chaque opération avec le magnétophone, mettre en marche celui-ci manuellement et l'arrêter ensuite.

Deux vitesses d'enregistrement/lecture peuvent être utilisées par Oric :

. 240 caractères par seconde (2400 bauds)

Cette vitesse nécessite des têtes de lecture/ écriture propres et biens alignées, et des cassettes de bonne qualité.

. 40 caractères par seconde (300 bauds) moins rapide, mais plus sûre.

- SAUVEGARDE ET CHARGEMENT D'UN PROGRAMME

Pour sauvegarder un programme sur cassette, mettez votre magnétophone en position d'enregistrement, tapez CSAVE "XX" et appuyez sur [RETURN]. (XX est le nom de votre programme, actuellement encore dans la mémoire vive ; ce nom peut compter jusqu'à 17 caractères y compris chiffres, points, barres obliques). Le message "SAVING XX" apparaît tout en haut de l'écran : ce qui veut dire que votre programme est bien en cours d'enregistrement. Lorsque c'est terminé, "Ready" s'affiche sur l'écran.

Puis, quelques jours plus tard vous désirez recharger "XX" dans Oric. Branchez alors correctement votre magnétophone et tapez : CLOAD "XX" après avoir mis votre magnétophone en position lecture. Oric cherche alors "XX" et pour l'indiquer affiche le message "SEARCHING". En haut de l'écran ce message se transforme en "LOADING XX".

Mais peut-être avez vous oublié le nom de votre programme ? Peu importe, recommencez la manipulation en tapant CLOAD "". Le programme chargé sera le premier trouvé, si ce n'est pas le bon, recommencez jusqu'à ce que ce soit le bon.

La commande CSAVE fait enregistrer à la vitesse de 2400 bauds. Si vous voulez soigner votre enregistrement dans le cas où votre magnétophone n'est plus de bonne qualité, vous devez enregistrer à 300 bauds. Si à 2400 bauds un problème survient, vous risquez de voir apparaître, au cours du chargement du programme, le message d'erreur : FILE ERROR - LOAD ABORTED (erreur de fichier, chargement abandonné). Pour éviter cela, lors de la sauvegarde, tapez : CSAVE "PROG1", S. PROG1 étant le nom du

programme, le S vient de slow (lent en anglais) et commande l'enregistrement à la vitesse de 300 bauds.

Pour les chargements de programmes sauvegardés à 300 bauds, tapez CLOAD "PROG1", S sinon Oric s'attendra à une vitesse de 2400 bauds. Si vous souhaitez faire exécuter votre programme automatiquement une fois qu'il a été chargé, tapez CSAVE "PROG1", AUTO.

Vous pouvez également sauvegarder des blocs mémoire. Pour cela vous devez connaître l'adresse, la localisation du début et de la fin de ce bloc. Ainsi :

CSAVE "PROGMEM", A #400, E #499

Cette instruction sauvera le bloc (appelé par vous PROGMEM) situé, rangé en RAM entre les localisations #400 et #499. Le signe # indique qu'il s'agit de nombres hexadécimaux. Le A et le E sont obligatoires : ils indiquent le début et la fin. Vous pouvez utiliser des nombres décimaux (sans le signe #).

Annexe D UTILISATION D'UNE IMPRIMANTE

Oric peut être utilisé avec toute imprimante ayant une interface Centronics. Pour cela, l'imprimante doit être connectée avant d'être mise en marche. Si tout est correct, la tête de l'imprimante doit s'aligner d'elle-même sur la position de départ dès que l'on alimente.

Dès que l'imprimante est branchée, tapez le programme suivant :

```
10 REM ** ESSAI D'IMPRESSION **
20 FOR N = 0 TO 255
30 LPRINT N, CHR$(N)
40 NEXT N
```

Tapez ensuite :

```
LLIST
```

Cela listera le programme sur l'imprimante au lieu de l'écran. Si vous n'obtenez pas des caractères "normaux", mais des signes inattendus, consultez la notice de l'imprimante pour changer les caractères d'impression, jusqu'à obtenir un listage satisfaisant.

Faites alors exécuter le programme ci-dessus qui est une simple boucle, imprimant un nombre suivi du caractère dont il est le code ASCII. Cela vous montrera quels caractères vous pouvez obtenir, et surtout quels nombres correspondent à des codes de contrôles pour l'imprimante. Les codes de contrôle commandent des actions telles que le retour du chariot, les caractères serrés ou non, petits ou grands, ou l'avance du papier, etc...

Par exemple essayez :

LPRINT CHR$(31) : les caractères doublent de taille.
LPRINT CHR$(30) : annule cette action.

REMARQUE : Les codes de contrôle peuvent donc vous être très utiles pour obtenir des impressions soignées, riches, utilisant des gadgets vous permettant d'obtenir des effets variés ou esthétiques.

Annexe E NUMERATION HEXADECIMALE,

REPRESENTATION DES REELS ET DES ENTIERS

1. LA NUMERATION HEXADECIMALE

Ainsi que vous l'avez vu dans les chapitres de présentation, la configuration d'un microordinateur est telle que les informations sont stockées dans des circuits intégrés.

La représentation physique de ces informations est donc un ensemble d'"interrupteurs" qui sont soit ouverts, soit fermés. L'état d'un interrupteur constitue une information élémentaire que l'on appelle bit ; un bit vaut donc soit 0, soit 1.

Chaque information est donc traduite en bits par la machine, c'est-à-dire en langage binaire. Pour plus de commodité technologique, ces bits sont réunis par paquets de huit que l'on appelle octets. L'octet est donc l'unité de base pour votre Oric.

Puisque Oric "travaille" sur des octets, il est donc commode d'utiliser une numération en tenant compte. C'est pourquoi on utilise une numération par paquets de quatre bits (par ensemble de seize états possibles).

A ce niveau il n'est pas inutile de rappeler les principes de la numération. Vous utilisez couramment la numération en base 10 (décimale) ; ceci n'ayant vraisemblablement de raison d'être que parce que nous possédons 10 doigts. En numération en base 10 nous utilisons 10 chiffres 0, 1, ..., 9. Chaque nombre, en base 10, ne s'écrit qu'à l'aide de ces 10 chiffres.

Par exemple, le nombre 7864 s'interprète comme

4 unités + 6 dizaines + 8 centaines + 7 milliers,

c'est-à-dire $(7864)_{base\ 10}$ = 4 + 6 x (base) + 8 x (base x base) + 7 x (base x base x base).

En numération en base 16 (ou hexadécimale), le principe est strictement le même. On utilise 16 chiffres : 0, 1, ..., 9, A, B, C, D, E, F

La traduction en base 10 de ce chiffre s'écrit :

$$\begin{array}{ll} (0)_{16} & = (0)_{10} \\ \vdots & \vdots \\ (9)_{16} & = (9)_{10} \\ (A)_{16} & = (10)_{10} \\ \vdots & \vdots \\ (F)_{16} & = (15)_{10} \end{array}$$

Ainsi le nombre A1E7 vaut 7 + E x (base) + 1 x (base x base) + A x (base x base x base),

soit $(A1E7)_{16}$ = 7 + 14 x (16) + 1 x (16 x 16) + 10 x (16 x 16 x 16) = $(41447)_{10}$

Le passage de la base décimale à la base hexadécimale s'effectue par l'opération inverse.

Considérons l'entier $(N)_{10}$. Pour obtenir son écriture en base 16, on divise $(N)_{10}$ par la valeur 16 jusqu'à obtenir un reste plus petit que la base. L'écriture s'effectue à l'aide de la valeur successive des $(restes)_{16}$.

Par exemple traduisons $(125342)_{10}$ en base 16 :

```
125 342 : 16
 13 3   : 7 833 : 16
  0 54  : 1 43  : 489 : 16
    062 :   153 :  09 : 30 : 16
     14 :    09 :    : 14 : 1 : 16
                           1 :  0
      E       9     9    E   1
```

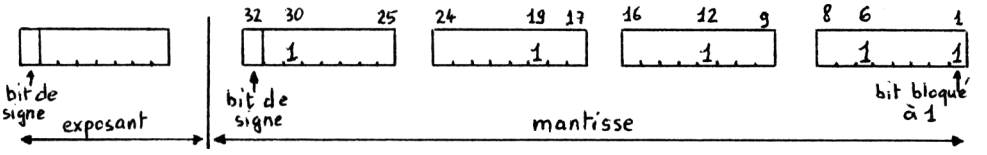
Donc $(125\ 342)_{10} = 14 + 9 \times (16) + 9 \times (16 \times 16) + 14 \times (16 \times 16 \times 16) + 1 \times (16 \times 16 \times 16 \times 16)$

$(125\ 342)_{10} = (1E99E)_{16}$

Il ne vous reste donc plus qu'à vérifier sur quelques exemples que vous avez compris le principe du passage d'une numération à l'autre. N'oubliez pas que votre Oric exécute facilement les conversions de la base décimale à la base hexadécimale et réciproquement.

2. LA REPRESENTATION DES REELS

Comme vous l'avez vu au chapitre 3 votre Oric stocke les réels sur cinq octets (un octet pour l'exposant et quatre pour la mantisse), voyons comment cela se passe physiquement.



. Pour l'exposant, il y a 7 bits ayant deux valeurs (0, 1) pour le représenter, puisque le dernier bit est "bloqué" par le signe. Donc pour représenter l'exposant il y a $2^7 = 128$ positions ($0 \rightarrow 127$, $-127 \rightarrow 0$).

. Puisque l'on raisonne en base deux, les exposants vont donc varier de :

$$2^{-127} \text{ à } 2^{127}$$

. La mantisse : elle est représentée sur 4 octets. Le dernier bit est le bit du signe et le premier bit est toujours bloqué à 1.

En fait, la mantisse est écrite en logarithme en base 2 : pour traduire en base hexadécimale le contenu d'une mantisse, on repère chaque bit valant 1 par sa position. (Ces bits sont appelés i et leur position p_i).

La mantisse vaut :

mantisse = somme $(2)^{-p_i}$ sur tous les bits

Ainsi, si on considère le contenu de la mantisse du schéma ci-dessus on a :

$$\text{mantisse} = 2^{-1} + 2^{-6} + 2^{-12} + 2^{-19} + 2^{-30}$$

donc la mantisse varie entre : $1/2 < \text{mantisse} < 1$
$-1/2 > \text{mantisse} > -1$

(On obtient 1/2 quand tous les bits du 2ème au 32ème sont à zéro).

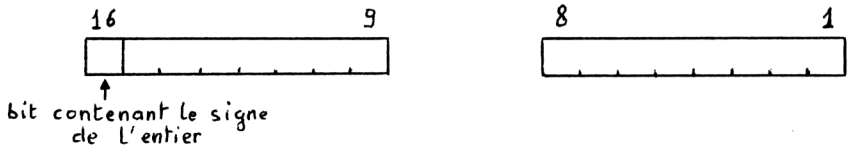
(On obtient 1 quand tous les bits du 2ème au 31ème sont à 1).

Donc, en valeur absolue $2^{-128} < \text{réel} < 2^{127}$

La mantisse est stockée sur 31 bits + bit de signe : la précision vaut $1/2 \times 2^{-31}$ (soit 10 chiffres significatifs).

3. REPRESENTATION DES ENTIERS

Vous avez vu au chapitre 3 que votre Oric stockait les entiers dans deux octets, voyons comment cela se traduit physiquement.



Il y a 15 bits pour stocker la valeur absolue d'un entier (valeur absolue (X) = Max (X;-X)), il y a donc 2^{15} positions possibles ; c'est pourquoi Oric ne les fait varier, en valeur absolue, qu'entre 0 et $2^{15} - 1$ = 32767.

ANNEXE F
Codes ASCII

ASCII Code	Caractère	ASCII Code	Caractère	ASCII Code	Caractère
000	NUL	036	$	072	H
001	SOH	037	%	073	I
002	STX	038	&	074	J
003	ETX	039	'	075	K
004	EOT	040	(	076	L
005	ENQ	041	)	077	M
006	ACK	042	*	078	N
007	BEL	043	+	079	O
008	BS	044	,	080	P
009	HT	045	-	081	Q
010	LF	046	.	082	R
011	VT	047	/	083	S
012	FF	048	0	084	T
013	CR	049	1	085	U
014	SO	050	2	086	V
015	SI	051	3	087	W
016	DLE	052	4	088	X
017	DC1	053	5	089	Y
018	DC2	054	6	090	Z
019	DC3	055	7	091	[
020	DC4	056	8	092	\
021	NAK	057	9	093	]
022	SYN	058	:	094	↑
023	ETB	059	;	095	£
024	CAN	060	<	096	©
025	EM	061	=	097	a
026	SUB	062	>	098	b
027	ESCAPE	063	?	099	c
028	FS	064	@	100	d
029	GS	065	A	101	e
030	RS	066	B	102	f
031	US	067	C	103	g
032	SPACE	068	D	104	h
033	!	069	E	105	i
034	"	070	F	106	j
035	#	071	G	107	k
108	l	115	s	122	z
109	m	116	t	123	{
110	n	117	u	124	|
111	o	118	v	125	}
112	p	119	w	126	~
113	q	120	x	127	DEL
114	r	121	y		

ASCII codes are in decimal.
LF=Line Feed, FF=Form Feed, CR=Carriage Return, DEL=Rubout

Annexe G CARTE MEMOIRE D'ORIC

MODE HIRES		MODE TEXT	
	FFFF		FFFF
ROM	C000	ROM	C000
ESPACE	BFE0	ESPACE	BFE0
ECRAN	A000	ECRAN	BB80
CARACTERES ALTERNES	9C00	CARACTERES ALTERNES	B800
CARACTERES STANDARD	9800	CARACTERES STANDARD	B400

Les adresses sont exprimées en hexadecimal ; elles peuvent suivre la commande POKE et vous permettre ainsi de contrôler une partie de l'écran

de BB80 à BFE0 (TEXT)
ou de A000 à BFE0 (HIRES)

ou d'accéder au stockage des caractères alternés ou standards pour en modifier les définitions par exemple.

9800 en décimal : 38912
9C00 en décimal : 39936
A000 en décimal : 40960
BFE0 en décimal : 49120
BB80 en décimal : 48000
B800 en décimal : 47104
B400 en décimal : 46080

Annexe H ROLE DES ATTRIBUTS

Décimal	Binaire	Hexadécimal	Effet
0	0	0	encre NOIRE
1	1	1	encre ROUGE
2	10	2	encre VERTE
3	11	3	encre JAUNE
4	100	4	encre BLEU
5	101	5	encre MAGENTA
6	110	6	encre CYAN
7	111	7	encre BLANCHE
8	1000	8	Caractère standard simple hauteur
9	1001	9	Caractère alterné simple hauteur
10	1010	A	Caractère standard double hauteur
11	1011	B	Caractère alterné double hauteur
12	1100	C	Clignotement simple hauteur standard
13	1101	D	Clignotement simple hauteur alterné
14	1110	E	Clignotement double hauteur standard
15	1111	F	Clignotement double hauteur alterné
16	10000	10	papier NOIR
17	10001	11	papier ROUGE
18	10010	12	papier VERT
19	10011	13	papier JAUNE
20	10100	14	papier BLEU
21	10101	15	papier MAGENTA
22	10110	16	papier CYAN
23	10111	17	papier BLANC
24	11000	18	Changement de synchronisation
31	1111	1F	

Les fonctions décrites dans ce tableau sont celles obtenues pour des attributs suivant les commandes PLOT, FILL, POKE.

L'utilisation de X, $0 \leqslant X \leqslant 31$ dans la commande PRINT CHR$(X) est plutôt à relier à l'action des caractères de contrôle (voir appendice B).

INDEX ALPHABETIQUE

A

ABS 93, 147
Acquisition de données 26
Addition 19, 71
Affectation 72
Alternée (forme) 103
AND 45, 73
Appel d'un sous-programme 90, 92
Arctangente..... 93
Arrêt d'une exécution de programme 31
ASC 94, 147
ASCII (code) 53, 187
ATN 93, 148
Attribut 104, 105, 113, 189

B

Basic 15
Basse résolution 105
Binaire (code) 13
Bit 13
Boucle 30, 51, 49, 83
BREAK 31
BYTE 13

C

CALL 92
Canal 137, 138, 139
Caractères alphanumériques 54
Caractères spéciaux 54
Carte mémoire 188
Cassette 7, 178
Cellule 105, 106
Chaîne de caractères 59
CHAR 117, 149
Choix conditionnel 43
CHR$ 94, 149
CIRCLE 118, 150
Clavier 2
CLEAR 39, 75, 150

Clignotement 113, 123, 125, 131
CLOAD 150
CLS 46
Code d'erreur 171
Commandes fondamentales 36, 145
Commandes graphiques haute résolution ... 108
Commandes graphiques basse résolution ... 114
Constantes 54, 57
CONT 31, 150
Correction dans un programme 40
COS 93, 151
Cosinus 93
Couleurs 108
Création de fonction utilisateur 95
CSAVE 152
CTRL A 42
CTRL C 31
CTRL D 125
CTRL F 135
CTRL G 136
CTRL L 47
CTRL X 20
CURMOV 115, 151
CURSET 114, 115, 151
Curseur 11
Curseur HIRES 114

D

DATA 99, 152
DEEK 152
DEF FN 96, 152
DEF FNA 96
DEF FNA$ 96
DEF USR 92, 153
Del 20
Dépassement de capacité 55
Différent (signe différent de) 72
DIM 153
Dimension d'un tableau 64
Division 21, 71
DOKE 153
Données 99
DRAW 115, 153
Durée du son 142

E

Effacement d'un programme 36
Effacement d'une ligne 41
Encre 108
END 40, 77, 154
Enregistrement d'un programme 178
Entrée/sortie 97
EXP 93, 154
Exponentielles 93
EXPLODE 50, 136, 154

F

FALSE 154
FILL 119, 154
FN 155
Fonctions 92
  numériques internes..... 92
  mathématiques simples 92
  mathématiques complexes 93
  traitant les chaînes de caractères 93
  de conversion ASCII/caractères 94
  de conversion chiffres/lettres 94

G

Génération d'un nombre aléatoire 92
GET 98, 156
GOSUB 91, 156
GOTO 29, 79, 156
GRAB 157
Graphiques 101
Graphiques avancés 127

H

Haute résolution 106
Hexadécimal (code) 56
HEX$ 157
HIMEM 157
HIRES 106, 157

I

IF ... THEN ... ELSE 43, 82, 158
ILLEGAL QUANTITY ERROR 81, 17
Indice 54
INK 108, 158
INPUT 26, 33, 97, 158
Insertion d'une ligne 40
Instruction 26
Instructions BASIC facultatives 75
Instructions de transfert 79
INT 48, 93, 158
Interface cassette 7, 178
Interface imprimante 181

J

Jeux 42

K

KEY$..... 98, 159

L

Langage 15, 53
LEFT$ 94, 159
LEN 95, 118, 159
LET 72, 78, 24, 159
LIST 30, 36, 159
Listage d'un programme 36
LLIST 160, 181
LN 160
LOG 93
Logarithme 93
LORES 103, 160

M

Mantisse 56
Mémoire 5
Microordinateur 1
Microprocesseur 1
MID$ 95,118, 161
Mise en route 8
MISMATCH ERROR 59
Mode graphique 102, 103
Mode direct 16
Mode différé 16
Modem 8
Modification d'une ligne 41
Mots réservés (ou clés) 25, 59, 145
MUSIC 138, 161

N

NEW 38, 161
NOT 74
Note 139

O

Octet 13
Opérateurs 22
  relationnels 72
  arithmétiques 71
  logiques 73
ON ... GOSUB 161, 91, 156
ON ... GOTO 162, 29, 79, 156
OR 46, 73
Ordres de transfert inconditionnel 79
Ordres de transfert conditionnel 82
OVERFLOW ERROR 55, 171

P

PAPER 162, 108
Papier 108

Partie entière 48
PATTERN 115, 162
PEEK 128, 162
Période 138
PI (signe mathématique Pi) 26, 59, 163
PING 51, 136, 163
Pixel 106, 107
PLAY 140, 141, 163
PLOT 108, 163
Point 119, 163
POKE 128, 163
POP 169
POS 164
PRINT 12, 16, 24, 29, 34, 164
Prise Péritel 7
Programme 18
Programmation 26
Puissance 21, 71
PULL 165

R

RAM 5
Racine carrée 93
READ 99, 165
RELEASE 165
Remplacement d'une ligne 28
REPEAT ... UNTIL 49, 83, 165
Remise à zéro des variables 75
REM 79, 165
Reprendre la main 89
RESET 89
RESTORE 32
Retour chariot 12
RETURN 12, 166
RIGHT$ 166, 194
RND 47, 166
ROM 18, 39, 166
RUN 18, 39, 166

S

SCRN 111, 112, 167
SGN 93, 167
SHIFT 12
SHOOT 51, 136, 167
Signe 93
SIN 93
Sinus 93
Sortie cassette 7, 178
Sortie vidéo 7
SOUND 137, 167
Sous-programme 89
Soustraction 19, 71
SPC 36, 168
SQR 31, 93, 168
STOP 40, 75, 168
STR$ 94, 168
Suppression d'une ligne..... 41
SYNTAX ERROR 12

T

TAB 168
Tableaux 64
  de variables entières 67
  de chaînes de caractères 68
  de variables réelles 69
TAN 93, 169
Tangente 93
TEXT 102, 169
Test 43
Traitement de données 26
TROFF 169
TRON 169
TRUE 169

U

UNDEF'D STATEMENT ERROR 80
Unité centrale 4
USR 170

V

VAL 94, 170
Valeur absolue 93
Variables simples 58
  entières 60
  réelles..... 62
  indicées 64
Volume 138, 140

W

WAIT 50, 170

Z

ZAP 51, 170

LA BIBLIOTHÈQUE EDIMICRO

- *Collection « Guides microordinateurs »*

de Merly	**GUIDE DE L'APPLE**
	Tome 1 L'APPLE standard
	Tome 2 Les extensions
	Tome 3 Les applications
	Tome 4 Les langages
Bayvejiel	**GUIDE DE L'ORIC**
Bieber, Perbost, Renucci	**GUIDE DU TO 7**

- *Collection « Jeux »*

Chane-Hune, Darbois	**JEUX SUR ORIC**
Perbost, Renucci	**JEUX SUR TO 7**

- *Collection « Logiciels »*

Bonnet, Dinh	**MULTIPLAN SUR APPLE**
	Exercices de Gestion

- *Ouvrages de base*

Viguier	**Premiers pas en programmation sur ORIC**

Les séminaires de formation FDS/Edimicro

- *Pour utilisateurs non spécialistes*

  VISICALC - MULTIPLAN - GESTION DE FICHIERS - BASES DE DONNÉES - CHOIX D'UN MICROORDINATEUR - BUREAUTIQUE...

- *Pour programmeurs débutants*

  PREMIERS PAS EN BASIC - FICHIERS EN BASIC...

- *Pour professionnels de l'informatique*

  TÉLÉMATIQUE - UNIX - XENIX - MS-DOS - LANGAGE C...

EDIMICRO
121-127, avenue d'Italie, 75013 PARIS

Imprimé en France. — JOUVE, 18, rue Saint-Denis, 75001 PARIS
Dépôt légal 1re édition : juin 1983
N° 11958. Dépôt légal : Novembre 1983

GUIDE DE L'ORIC

Philippe Bayvejiel

« Débutants ou experts, amateurs de jeux ou professionnels de la gestion, ce livre vous passionnera. Laissez-vous conduire pas-à-pas, et vous constaterez à quel point il est facile d'utiliser et de programmer votre ORIC ».

Denis TAIEB
Importateur de l'ORIC en France

L'ORIC se place dans le peloton de tête des ordinateurs personnels de sa catégorie : dès son apparition, toute récente, le public lui a fait un excellent accueil.

Ce guide s'adresse à tous les possesseurs et utilisateurs de l'ORIC, ainsi qu'à toutes les personnes qui envisagent d'acheter un ordinateur familial et professionnel. Chacun trouvera ici un bon exemple de ce que peut faire un petit ordinateur et apprendra à s'en servir même s'il n'a aucune connaissance en informatique.

Le GUIDE DE L'ORIC adopte une démarche très progressive, illustrée par de nombreux exemples :

Chapitre 1 : présentation générale de l'ORIC.
Chapitre 2 : initiation à la programmation ; créez votre jeu en BASIC.
Chapitre 3 : Le BASIC de l'ORIC.
Chapitre 4 : possibilités graphiques de l'ORIC.
Chapitre 5 : les sons et l'ORIC.

Edimicro

GUIDE DE L'ORIC

Philippe Bayvejiel